TECHNICAL UNIVERSITY OF CRETE
Department of Electronic and Computer Engineering

Maximization of a rank-4 quadratic form by a binary vector with complexity $\mathcal{O}(N^3 \log N)$

Submitted to the Department of Electronic and Computer Engineering in partial fulfillment of the requirements for the ECE Diploma Degree.

By: Alexandros Sklikas

Advisor: Associate Professor George Karystinos
Committee Member: Associate Professor Aggelos Bletsas
Committee Member: Assistant Professor Vasilis Samoladas

Abstract

We consider the problem of maximizing a quadratic form over the binary alphabet. This problem is known as the unconstrained $(-1, 1)$-quadratic maximization problem or binary quadratic programming (in computer science terminology) and is an NP-hard combinatorial problem that can be solved through an exponential-complexity exhaustive search. Recently, it has been shown that the exhaustive search is not necessary and this problem is polynomially solved, if the rank of the quadratic form is constant (which is a case that is met is certain optimization problems in communication theory). A few polynomial-time algorithms have been reported from several research groups that differ in their actual space and/or time complexity.

In this thesis, we focus on the case where the rank of the form is 4 and present an optimal algorithm with complexity $O(N^3\log(N))$ that is based on novel ideas that combine the auxiliary-angle framework developed in TUC and a few elements from computational geometry. For completeness, we present our method for the cases of rank-2 and rank-3 quadratic forms, with complexity $O(N\log(N))$ and $O(N^2\log(N))$, respectively. For all three cases, we show that our algorithm is the fastest known implementable one among the several choices in the literature. Finally, we also comment on how our approach can be generalized to any rank-D quadratic form and lead to an algorithm of complexity $O(N^{D-1}\log(N))$.

Acknowledgements

First of all I would like to thank my parents. Without your continued and unyielding support the fulfilment of this thesis would not be possible.

I would also like to thank my mentor and advisor, professor George Karystinos for his guidance and assistance in every step of this thesis.

Finally, I would like to thank all my friends at the Technical University of Crete for all the great times we have had together and my dear Sofia who has always been there for me.

Contents

Chapter 1

Problem Statement

We consider a quadratic form

$$\mathbf{s}^T\mathbf{A}\mathbf{s} \tag{1.1}$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a positive (semi)definite matrix parameter and $\mathbf{s} \in \mathbb{R}^N$ is a vector argument. The complexity of the maximisation of $\mathbf{s}^T\mathbf{A}\mathbf{s}$ with respect to $\mathbf{s}$, subject to a norm constraint on $\mathbf{s}$, depends on the feasible set $\mathcal{S}$ of $\mathbf{s}$ as well as the characteristics of $\mathbf{A}$. If $\mathcal{S} = \{\mathbf{s} \in \mathbb{R}^N : \|\mathbf{s}\| = c > 0\}$, then $\mathbf{s}^T\mathbf{A}\mathbf{s}$ is maximised by the appropriate scaled maximum-eigenvalue eigenvector of $\mathbf{A}$. If now $\mathcal{S} = \{\pm 1\}^N$, then

$$\mathbf{s}_{opt} \triangleq \arg \max_{\mathbf{s}\in\{\pm 1\}^N} \mathbf{s}^T\mathbf{A}\mathbf{s} \tag{1.2}$$

widely knowns as the unconstrained $(-1, 1)$-quadratic maximisation problem or binary quadratic programming, becomes NP-hard for an arbitrary matrix $\mathbf{A}$ and can be computed by exhaustive search among all elements of $\mathcal{S}$ with complexity $\mathcal{O}(2^N)$ since $|\mathcal{S}| = 2^N$, an approach that becomes intractable even for moderate values of N.

The above problem has a wide variety of applications including the adaptive design of binary spreading codes [1], maximum likelihood (ML) coherent MIMO detection and ML non-coherent SIMO detection [2], statistical physics and circuit design [3], [4], [5] and limited-feedback MIMO beamforming [6], [7], [8], [9], [10], [11], [12], [13]. Furthermore, integer-least-squares optimization [14], [15] is equivalent to positive semi-definite quadratic form maximization when the vector argument is binary. As a result multiuser detection in CDMA [16] [17], or multicarrier CDMA systems [18], lattice-code decoding, [19], [20], [21], soft decision decoding, [22], [23], [24], linear-dispersive [25], lattice-code [26] and quasi-orthogonal [27] space-time block-code (STBC) decoding, signal detection upon frequency-selective finite-impulse-response channel processing [28], peak-to-average-power

ratio reduction in orthogonal frequency-division multiplexing (OFDM) systems [29], joint detection and channel estimation in OFDM systems [30], equalization of block transmissions [31] and signal detection in general multi-antenna MIMO systems [32] are examples of optimization problems that involve positive (semi)definite quadratic form maximization with a binary vector argument.

Since $\mathbf{A}$ is positive (semi)definite,

$$\mathbf{A} = \sum_{i=1}^{N} \lambda_i \mathbf{q}_i \mathbf{q}_i^T, \ \lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_N \geqslant 0, \ \|\mathbf{q}_i\| = 1 \tag{1.3}$$

represents its eigendecomposition where λ_i and $\mathbf{q}_i$ are the ith eigenvalue and normalised eigenvector, respectively, of $\mathbf{A}$. Now, if $\mathbf{A}$ is rank deficient, then an exhaustive search over $\mathcal{S} = \{\pm 1\}^N$ is not necessary. For example, if $\mathbf{A}$ is of rank 1, then $\mathbf{s}_{opt}$ in (1.2) can be derived by inspection and is given by $\mathbf{s}_{opt} = sgn(\mathbf{q}_1)$, where $sgn(\cdot)$ is the vector sign operation.[1]

Matrix $\mathbf{A}$ of rank D can be eigendecomposed as

$$\mathbf{A} = \sum_{i=1}^{D} \lambda_i \mathbf{q}_i \mathbf{q}_i^T, \tag{1.4}$$

where $\lambda_i > 0, i = 1, 2, \ldots, D$. We define the weighted eigenvectors $\mathbf{v}_i \triangleq \sqrt{\lambda_i}\mathbf{q}_i$, $i = 1, 2, \ldots, D$ and concatenate them to form the $N \times D$ matrix $\mathbf{V} \triangleq [\mathbf{v}_1 \ \mathbf{v}_2 \ \ldots \mathbf{v}_D]$. Then the rank-D quadratic form becomes

$$\begin{aligned} \mathbf{s}^T\mathbf{A}\mathbf{s} &= \mathbf{s}^T(\lambda_1 \mathbf{q}_1\mathbf{q}_1^T + \lambda_2 \mathbf{q}_2\mathbf{q}_2^T + \cdots + \lambda_D \mathbf{q}_D\mathbf{q}_D^T)\mathbf{s} \\ &= (\mathbf{s}^T\mathbf{v}_1)^2 + (\mathbf{s}^T\mathbf{v}_2)^2 + \cdots + (\mathbf{s}^T\mathbf{v}_D)^2 = \|\mathbf{V}^T\mathbf{s}\|^2. \end{aligned} \tag{1.5}$$

Finally, $\mathbf{s}_{opt}$ defined in (1.2) maximises also the square root of the quadratic form, we turn our interest into the equivalent problem

$$\max_{\mathbf{s}\in\{\pm 1\}^N} \sqrt{\mathbf{s}^T\mathbf{A}\mathbf{s}} = \max_{\mathbf{s}\in\{\pm 1\}^N} \sqrt{\|\mathbf{V}^T\mathbf{s}\|^2} = \max_{\mathbf{s}\in\{\pm 1\}^N} \|\mathbf{V}^T\mathbf{s}\|. \tag{1.6}$$

[1] For any $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{y} = sgn(\mathbf{x})$ is an $N \times 1$ vector with $y_l = \begin{cases} -1 & x_l < 0, \\ 1 & x_l > 0, \end{cases}$, $l = 1, 2, \ldots, N$.

Chapter 2

A summary of approaches

Solving (1.6) when $\mathbf{A}$ is a symmetric real matrix $N \times N$ is an NP-hard problem, though some polynomially solvable cases do exist. The following table provides information concerning the time and space complexity of the methods, as well as the size of the candidate set $\mathcal{S}$, when $\mathbf{A}$ is *positive semi-definite and rank($\boldsymbol{A}$)= D (constant).*

Algorithm	Size of Candidate Set $\mathcal{S}$	Time Complexity	Space Complexity
[37], [38], [39]	$\mathcal{O}(N^{D-1})$	$\mathcal{O}(N^{D-1})$	$\mathcal{O}(N^{D-1})$
[44]	$\mathcal{O}(N^{D-1})$	$\mathcal{O}(N^{D-1}\log N)$	$\mathcal{O}(N^{D-1})$
[1], [41] (D=2, 3)	$\mathcal{O}(N^{D-1})$	$\mathcal{O}(N^{D-1}\log N)$	$\mathcal{O}(N^{D-1})$
[1], [2], [42] ($D \geqslant 2$)	$\mathcal{O}(N^{D-1})$	$\mathcal{O}(N^{D})$	$\mathcal{O}(N)$
[43]	$\mathcal{O}(N^{D-1})$	$\mathcal{O}(N^{D})$	
[33], [34], [35]	$\mathcal{O}(N^{D-1})$	$\mathcal{O}(N^{D+1})$	$\mathcal{O}(N^{R})$

The above case was proven in [33] and an algorithm was created in [33], [34] based on the reverse search in [35]. This algorithm creates a set of candidates $\mathcal{S}$, which includes the solution to (1.6), of size $\mathcal{O}(N^{D-1})$. This has a time complexity of $\mathcal{O}(N^{D}\mathbf{LP}(N, D))$ where $\mathbf{LP}(N, D)$ is the time required to solve a linear problem with N inequalities in D variables. Work in [36] shows that $\mathbf{LP}(N, D) = \mathcal{O}(N)$, making the overall time complexity of the reverse search based algorithm $\mathcal{O}(N^{D+1})$.

Another approach to the solution of (1.6) was presented in [37]. The algorithm presented there is based on the incremental algorithm for cell enumeration in arrangements [38], [39], and also has polynomial time complexity. Despite its time efficient(overall complexity $\mathcal{O}(N^{D-1})$) construction of the set $\mathcal{S}$ whose size is $|\mathcal{S}| = \mathcal{O}(N^{D-1})$, this solution becomes impractical as rank($\mathbf{A}$)= D increases. Even for moderate values of D, the *incremental* construction of $\mathcal{S}$ requires memory proportional to $|\mathcal{S}|$.

Following a different perspective, one we also analyse in-depth in the following chapters and adhere to in our solution, and based on the auxiliary-angle approach originally introduced in [40], efficient solutions for (1.6) have been proposed for D $\geqslant$ 2. [1] solves the D=2 case, [41] the D=3 one and [2] the D>3, with complexity of $\mathcal{O}(N\log N), \mathcal{O}(N^2\log N)$ and $\mathcal{O}(N^D)$ respectively. The algorithm uses $D-1$ auxiliary angles to partition the $(D-1)$-dimensional hypercube into a set of distinct regions of polynomial size. Each region is associated with a unique binary vector, while the set of binary vectors produced has the same size with the one created by the reverse search methodology, $|\mathcal{S}|$.

[40], [37], [42], [43] produce polynomial time solutions when $\mathbf{s}$ is an MPSK vector and $\mathbf{A}$ is a hermitian positive semi-definite matrix of rank D. This includes our original case in this chapter as a special case. More specifically, [40] introduces an algorithm for solving the D=1 case with $\mathcal{O}(N\log N)$ while [37], [42], [43] solve for $D \geqslant 1$ with $\mathcal{O}(N^{2D})$.

The authors of [44] present an algorithm for solving the following two cases:

1. rank($\mathbf{A}$)= D (constant) and $A_{ii} \geqslant 0$, for any $i \in \{1, 2, \ldots, N\}$.

2. rank($\mathbf{A}$)= D (constant) and $|\{i : A_{ii} < 0\}| = \mathcal{O}(\log N)$.

with polynomial complexity. 2. includes 1. as a special case, while 1. includes the original case at the start of the chapter as a special one. More specifically, [44] presents an algorithm for solving the case in 1. by constructing a set $\mathcal{S}$ of size $|\mathcal{S}| = \mathcal{O}(N^{D-1})$ that includes the solution to (1.6) with a time complexity of $\mathcal{O}(N^{D-1}\log N)$ and a space complexity bound by the output size $\mathcal{O}(N^{D-1})$. Finally [44] shows the polynomial solvability of 2.

In a later chapter, we also focus on this (recursive) approach for solving (1.6). We present and analyse the algorithm and its complexity and compare it to the our own which makes use of auxiliary spherical coordinates.

Chapter 3

Introducing spherical coordinates

Let us now focus on

$$\max_{\mathbf{s}\in\{\pm 1\}^N} \|\mathbf{V}^T\mathbf{s}\|. \tag{3.1}$$

We recall that $\mathbf{V}$ is a full-rank $N \times D$ matrix, $D \leqslant N-1$. W.l.o.g. we assume that each row of $\mathbf{V}$ has at least one nonzero element, i.e. $\mathbf{V}_{l,1:D} \neq \mathbf{0}_{1:D}$. Indeed, if there exists an index $\in \{1, 2, \ldots, N\}$ such that $\mathbf{V}_{l,1:D} = \mathbf{0}_{1:D}$ then neither $s_n = +1$ nor $s_n = -1$ have an effect on $\mathbf{V}^T\mathbf{s}$ in (3.1), implying that we can ignore the corresponding row of $\mathbf{V}$, assign an arbitrary value to $s_n = \pm 1$, and reduce the size of the original problem from N to $N-1$. In addition, w.l.o.g. we assume that $V_{n,1} \neq 0, n = 1, 2, \ldots, N$, because for any $\mathbf{V} \in \mathbb{R}^{N\times D}$ there exists an orthogonal matrix $\mathbf{U} \in \mathbb{R}^{D\times D}$ such that $\|\mathbf{V}^T\mathbf{s}\| = \|(\mathbf{VU})^T\mathbf{s}\|$ and the $N \times D$ matrix $\mathbf{VU}$ contains no zero in its first column, i.e. $[\mathbf{VU}]_{n,1} \neq 0, n = 1, 2, \ldots, N$.

3.1 Rank-2 case

In order to solve the $D = 2$ case, we introduce a single spherical coordinate $\boldsymbol{\phi} \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and a hyperpolar vector $\mathbf{c}(\boldsymbol{\phi}) = [\sin(\phi),\ \cos(\phi)]$. Thus, and by using the Cauchy-Swartz inequality, the maximisation of $\|\mathbf{V}^T\mathbf{s}\|$ becomes the equivalent:

$$\begin{aligned} \max_{\mathbf{s}} \|\mathbf{V}^T\mathbf{s}\| &= \max_{\mathbf{s}}\ \max_{\phi}\{\mathbf{c}^T(\boldsymbol{\phi})\mathbf{V}^T\mathbf{s}\} \\ &= \max_{\phi}\ \sum_{n=1}^{N} \max_{\mathbf{s}}\{s_n\mathbf{V}_{n,:}\mathbf{c}(\boldsymbol{\phi})\}. \end{aligned} \tag{3.2}$$

Each term s_n of the binary vector $\mathbf{s}$ is given by

$$s_n(\boldsymbol{\phi}) = \begin{cases} +1,\ \mathbf{V}_{n,:}\mathbf{c}(\boldsymbol{\phi}) > 0 \\ -1,\ \mathbf{V}_{n,:}\mathbf{c}(\boldsymbol{\phi}) < 0 \end{cases}, n = 1, 2, \ldots, N \tag{3.3}$$

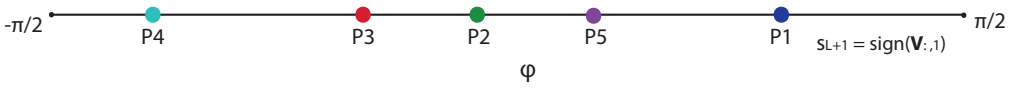


Figure 3.1: A rank-2 example with $N = 5$

and its value ± 1 is defined solely by the sign of $\mathbf{V}_{n,:}\mathbf{c}(\boldsymbol{\phi})$. (3.3) is equivalent to

$$s_n(\boldsymbol{\phi}) = \begin{cases} -sgn(\mathbf{V}_{n,1}), & \phi \in (-\frac{\pi}{2}, \tan^{-1}(-\frac{\mathbf{V}_{n,2}}{V_{n,1}})) \\ sgn(\mathbf{V}_{n,1}), & \phi \in (\tan^{-1}(-\frac{\mathbf{V}_{n,2}}{V_{n,1}}), \frac{\pi}{2}] \end{cases} \tag{3.4}$$

which in turn allows us to define the function

$$\phi_n \triangleq \tan^{-1}(-\frac{\mathbf{V}_{n,2}}{\mathbf{V}_{n,1}}) \in (-\frac{\pi}{2}, \frac{\pi}{2})$$

that partitions the 1-dimensional space into two regions with opposite values ± 1. In the rank-2 case, the function ϕ_n defines points on the 1-dimensional space.

Figure 3.1 shows an example of the rank-2 scenario. For $N = 5$ we can see that the function ϕ_n defines 5 points on ϕ, each of which divides the 1-D space into two regions with opposite values ± 1. While ϕ changes value, we can see the the regions that were created can be associated with a unique binary vector that does not change value while ϕ is in that region. By utilising this observation we can give a brief explanation of the workings of the serial and parallel algorithms.

First we need to calculate and sort all points ϕ_n. Then the serial approach can use the binary vector associated with the right-most region as a starting point. Since this region is "right" of all the points ϕ_n, according to (3.4) its value is equal to $sign(\mathbf{V}_{n,1})$, i.e. the sign of the values of the first column of $\mathbf{V}$. Then by iterating over all points ϕ_n and changing the sign of the corresponding element, the serial algorithm produces all the necessary binary vectors. The parallelisable approach on the other hand, visits each point ϕ_n independently to compute the value of the corresponding element in the candidate binary vectors. By doing this it is able to keep the space complexity rank-independent, $\mathcal{O}(N)$. This can become a deciding factor as D grows (remember that the space complexity of the serial algorithm is $\mathcal{O}(N^{D-1})$. Note also that the serial approach is vulnerable to error propagation. If at some point an entry s_n of a candidate vector is erroneously computed, then this mistake will continue to have an effect on all candidates that are created thereafter.

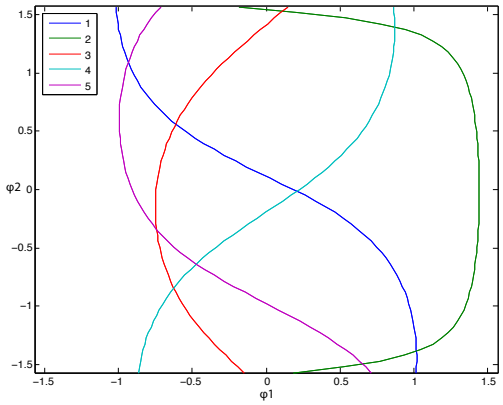


Figure 3.2: A rank-3 example with $N = 5$ curves

3.2 Rank-3 case

By increasing the rank D to 3, we need to introduce two spherical coordinates, $\boldsymbol{\phi_1}, \boldsymbol{\phi_2} \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, while the hyperpolar vector in this instance becomes

$$c(\boldsymbol{\phi_1}, \boldsymbol{\phi_2}) = [sin(\phi_1),\ cos(\phi_1)sin(\phi_2),\ cos(\phi_1)cos(\phi_2)].$$

Once again, by using the Cauchy-Swartz inequality, the maximisation of $\|\mathbf{V}^T\mathbf{s}\|$ becomes the equivalent:

$$\begin{aligned} \max_{\mathbf{s}} \|\mathbf{V}^T\mathbf{s}\| &= \max_{\mathbf{s}}\ \max_{\phi}\{\mathbf{c}^T(\boldsymbol{\phi})\mathbf{V}^T\mathbf{s}\} \\ &= \max_{\phi}\ \sum_{n=1}^{N} \max_{\mathbf{s}}\{s_n\mathbf{V}_{n,:}\mathbf{c}(\boldsymbol{\phi})\} \end{aligned} \tag{3.5}$$

and each term s_n of the binary vector $\mathbf{s}$ is given by

$$s_n(\boldsymbol{\phi}) = \begin{cases} +1,\ \mathbf{V}_{n,:}\mathbf{c}(\boldsymbol{\phi}) > 0 \\ -1,\ \mathbf{V}_{n,:}\mathbf{c}(\boldsymbol{\phi}) < 0 \end{cases}, n = 1, 2, \ldots, N \tag{3.6}$$

and its value ± 1 is only defined by the sign of $\mathbf{V}_{n,:}\mathbf{c}(\boldsymbol{\phi})$. (3.6) is equivalent to

$$s_n(\boldsymbol{\phi}) = \begin{cases} -sgn(\mathbf{V}_{n,1}), & \phi \in (-\frac{\pi}{2}, \tan^{-1}(-\frac{\mathbf{V}_{n,2}\sin(\phi_2)+\mathbf{V}_{n,3}\cos(\phi_2)}{V_{n,1}})) \\ sgn(\mathbf{V}_{n,1}), & \phi \in \tan^{-1}(-\frac{\mathbf{V}_{n,2}\sin(\phi_2)+\mathbf{V}_{n,3}\cos(\phi_2)}{V_{n,1}}), \frac{\pi}{2}] \end{cases} \tag{3.7}$$

which in turn allows us to define the function

$$\phi_n \triangleq \tan^{-1}(-\frac{\mathbf{V}_{n,2}\sin(\phi_2) + \mathbf{V}_{n,3}\cos(\phi_2)}{V_{n,1}}) \in (-\frac{\pi}{2}, \frac{\pi}{2})$$

Note that the function ϕ_n has changed and now defines a curve on the 2-dimensional space defined by ϕ_1, ϕ_2. Despite this, its functionality remains the same: it divides the 2-dimensional space into two regions with opposite values ± 1.

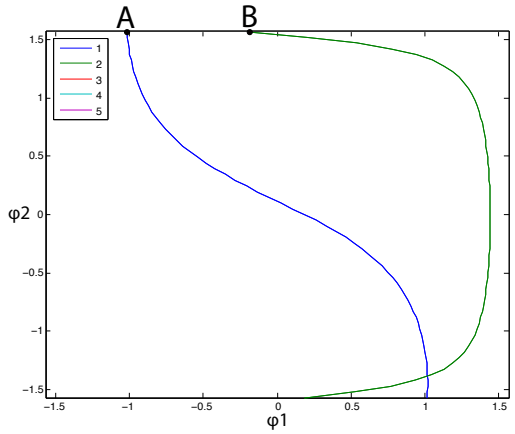


Figure 3.3: Solving the ambiguity at an intersection

Figure 3.2 depicts an example of a rank-3 scenario where $N = 5$. The function ϕ_n defines 5 curves creating regions with opposite vectors. Note also that the curves ϕ_n intersect two at a time creating a series of intersections. It is shown in [2] that the polarity of the intersection with regard to all curves is the same as the polarity of all points of the cell that lies "above" it. Also, each cell is associated with a unique candidate vector (given by the polarity) that remains unchanged for all points in the cell. It is sufficient to search among the candidate vectors in this arrangement in order to find the optimal one. All elements of the binary vector that is associated with a cell are well defined at an intersection, with the exception of the two elements associated with the intersecting curves.

The execution of the parallel algorithm remains the same. By visiting each intersection independently it is able to calculate the necessary binary vectors and continues to keep a low space complexity of $\mathcal{O}(N)$. The ambiguity that arises for the intersecting curves is solved at $\boldsymbol{\phi}_2 = \frac{\pi}{2}$ as can be seen in figure 3.3. Point A yields the sign for curve ϕ_2 (green) while point B the sign for ϕ_1 (blue).

On the other hand the serial algorithm starts by executing a rank-2 case at $\boldsymbol{\phi}_2 = -\frac{\pi}{2}$ and saves all $N + 1$ candidate vectors. This set of $N + 1$ candidates is continuously kept and updated throughout the execution of the algorithm. By moving along $\boldsymbol{\phi}_2$ towards $\frac{\pi}{2}$, it visits each intersection in an iterative fashion. At each intersection, the algorithm finds the cell that lies "beneath" the intersection. Then by only *flipping* the sign of the elements of its binary vector that are associated with the two intersecting curves, it produces the candidate that is associated with the cell that lies above the intersection. The set of $N+1$ candidates is updated and the algorithm continues on to the next intersection until it has calculated all possible candidates.

Once again the serial algortihm suffers for the probability of error propagation, something that the parallel algorithm avoids, though at the cost of higher complexity. Another problem arises for the serial algorithm that directly affects its time complexity; the reason that it would be preferred over the parallel one. A serial search among the $N+1$ elements of the saved set would have a complexity of $\mathcal{O}(N)$. By doing this for all $\binom{N}{2} = \mathcal{O}(N^2)$ intersections we only achieve a complexity of $\mathcal{O}(N^3)$, which is larger than the theoretic bound of $\mathcal{O}(N^2 \log N)$. The solution to this problem that arises during the implementation of the serial algorithm is the starting point, and one of the innovations we introduce in the next chapter where we present our approach to solving the problem.

A problem with the serial approach

3.3 Rank-D case

In this section we generalise the ideas presented in the rank-2 and rank-3 cases above into a unified form, for the general rank-D case. For completeness, we also present the parallel approach for solving this general case.

To develop an efficient method for the maximisation in (3.1), we introduce the spherical coordinates $\phi_1 \in (-\pi, \pi], \phi_2, \ldots, \phi_{D-1} \in (-\frac{\pi}{2}, \frac{\pi}{2}]$ and define the spherical coordinate vector

$$\boldsymbol{\phi}_{i,j} \stackrel{\Delta}{=} [\phi_i, \phi_{i+1}, \ldots, \phi_j]^T \tag{3.8}$$

and the hyperpolar vector

$$\mathbf{c}(\phi_{1:D-1}) \stackrel{\Delta}{=} \begin{bmatrix} \sin\phi_1 \\ \cos\phi_1 \ \sin\phi_2 \\ \cos\phi_1 \ \cos\phi_2 \ \sin\phi_3 \\ \vdots \\ \cos\phi_1 \ \cos\phi_2 \ \ldots \ \sin\phi_{D-1} \\ \cos\phi_1 \ \cos\phi_2 \ \ldots \ \cos\phi_{D-1} \end{bmatrix}. \tag{3.9}$$

A critical equality for our subsequent developments is

$$\max_{s\in\{\pm1\}^N} \|\mathbf{V}^T\mathbf{s}\| = \max_{s\in\{\pm1\}^N} \max_{\phi_{1:D-1}\in(-\pi,\pi]\times(-\frac{\pi}{2},\frac{\pi}{2}]^{D-2}} \{\mathbf{s}^T\mathbf{V}\mathbf{c}(\boldsymbol{\phi}_{1:D-1})\} \tag{3.10}$$

which results from the Cauchy-Schwartz Inequality, since for any $\mathbf{a} \in \mathbb{R}^D$

$$\mathbf{a}^T\mathbf{c}(\boldsymbol{\phi}_{1:D-1}) \leqslant \|\mathbf{a}\| \underbrace{\|\mathbf{c}(\boldsymbol{\phi}_{1:D-1})\|}_{=1} \tag{3.11}$$

with equality if and only if $\phi_1, \ldots, \phi_{D-1}$ are the spherical coordinates of $\mathbf{a}$. Interchanging the maximisations in (3.10) we obtain the equivalent problem

$$\max_{\phi_{1:D-1}\in(-\pi,\pi]\times(-\frac{\pi}{2},\frac{\pi}{2}]^{D-2}} \sum_{n=1}^{N} \max_{s_n=\pm 1} \{s_n \mathbf{V}_{n,1:D}\mathbf{c}(\boldsymbol{\phi}_{1:D-1})\}. \tag{3.12}$$

For fixed spherical coordinates $\boldsymbol{\phi}$ the maximising argument of each term of the sum in (3.12) is defined as

$$s_l(\boldsymbol{\phi}_{1:D-1}) \triangleq \arg \max_{s_n=\pm 1} s_n \mathbf{V}_{n,1:D}\mathbf{c}(\boldsymbol{\phi}_{1:D-1}),\ l = 1, 2, \ldots, N, \tag{3.13}$$

and is determined only by

$$\mathbf{V}_{n,1:D}\mathbf{c}(\boldsymbol{\phi}_{1:D-1}) \gtrless_{s_n(\boldsymbol{\phi}_{1:D-1})=-1}^{s_n(\boldsymbol{\phi}_{1:D-1})=+1} 0,\ l = 1, 2, \ldots, N. \tag{3.14}$$

For any $\boldsymbol{\phi}$, the corresponding optimal binary vector is defined as

$$\mathbf{s}(\boldsymbol{\phi}_{1:D-1}) \triangleq [s_1(\boldsymbol{\phi}_{1:D-1}) s_2(\boldsymbol{\phi}_{1:D-1}) \ldots s_N(\boldsymbol{\phi}_{1:D-1})]^T. \tag{3.15}$$

We note that $\mathbf{s}(\boldsymbol{\phi}_1 - \pi, \boldsymbol{\phi}_{2:D-1}) = -\mathbf{s}(\boldsymbol{\phi}_1, \boldsymbol{\phi}_{2:D-1})$, because $\mathbf{c}(\boldsymbol{\phi}_1 - \pi, \boldsymbol{\phi}_{2:D-1}) = -\mathbf{c}(\boldsymbol{\phi}_1, \boldsymbol{\phi}_{2:D-1})$ for any $\boldsymbol{\phi}_{1:D-1} \in (-\pi, \pi] \times (-\frac{\pi}{2}, \frac{\pi}{2}]^{D-2}$. Since opposite binary vector $\mathbf{s}$ and $-\mathbf{s}$ result in the same value of the quadratic form in (1.2), it suffices to search over $\boldsymbol{\phi} \in \Phi^{D-1}$ where $\Phi \triangleq (-\frac{\pi}{2}, \frac{\pi}{2}]$. Finally we collect all possible vectors $s(\boldsymbol{\phi}_{1:D-1})$ in the set

$$\begin{aligned} \mathcal{S}' &\triangleq \bigcup_{\boldsymbol{\phi}_{1:D-1}\in\Phi^{D-1}} \{s(\boldsymbol{\phi}_{1:D-1})\} \\ &= \{\mathbf{s} \in \{\pm 1\}^N : \forall \boldsymbol{\phi}_{1:D-1} \in \Phi^{D-1} \text{such that} \mathbf{s}(\boldsymbol{\phi}_{1:D-1}) = \mathbf{s}\} \end{aligned} \tag{3.16}$$

and note that $\mathbf{s}_{opt} \in \mathcal{S}'$. Returning to (3.14) we find that

$$s_n(\boldsymbol{\phi}_{1:D-1}) = \begin{cases} -sgn(\mathbf{V}_{n,1}), & \phi \in (-\frac{\pi}{2}, \tan^{-1}(-\frac{\mathbf{V}_{n,2:D}\mathbf{c}(\boldsymbol{\phi}_{2:D-1})}{V_{l,1}})) \\ sgn(\mathbf{V}_{n,1}), & \phi \in (\tan^{-1}(-\frac{\mathbf{V}_{n,2:D}\mathbf{c}(\boldsymbol{\phi}_{2:D-1})}{V_{n,1}}), \frac{\pi}{2}] \end{cases} \tag{3.17}$$

We define the function

$$\phi_n(\boldsymbol{\phi}_{2:D-1}) \triangleq \tan^{-1}\left(-\frac{\mathbf{V}_{n,2:D}\mathbf{c}(\boldsymbol{\phi}_{2:D-1})}{\mathbf{V}_{n,1}}\right) \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right) \tag{3.18}$$

that is equivalent to $\mathbf{V}^T\mathbf{c}(\boldsymbol{\phi}_{1:D-1}) = 0$, and stands since $\mathbf{V}_{n,1} \neq 0,\ l = 1, 2, \ldots, N$. The hypersurface $(\phi_n(\boldsymbol{\phi}_{2:D-1}), \boldsymbol{\phi}_{2:D-1})$ partitions Φ^{D-1} into two regions that correspond to the opposite values $s_n(\boldsymbol{\phi}_{1:D-1}) = \pm 1$. By creating all N hypersurfaces we effectively partition the domain Φ^{D-1} into a set of *cells* each of which defines a unique binary vector $\mathbf{s}_c(\boldsymbol{\phi}_{1:D-1})$. This set of vectors is sufficient for the maximisation of the quadratic form.

A parallelizable approach for solving rank-D

In this section we will discuss the approach presented in [2]. The authors show that each subset of $D-1$ hypersurfaces has either a single intersection or uncountably many intersections in Φ^{D-1} and also that the following hold true:

(i) $\mathbf{s}(\mathbf{V}_{N\times D};\phi_{1:D-2},\frac{\pi}{2}) = \mathbf{s}(\mathbf{V}_{N\times(D-1)};\phi_{1:D-2})$,

(ii) $\mathbf{s}(\mathbf{V}_{N\times D};\phi_{1:D-2},-\frac{\pi}{2}) = -\mathbf{s}(\mathbf{V}_{N\times D};-\phi_{1:D-2},\frac{\pi}{2})$,

(iii) $\mathbf{s}(\mathbf{V}_{N\times D};\phi_{1:D-3},\frac{\pi}{2},\phi_{D-1}) = \mathbf{s}(\mathbf{V}_{N\times(D-2)};\phi_{1:D-3})$,

(iv) $\mathbf{s}(\mathbf{V}_{N\times D};\phi_{1:D-3},-\frac{\pi}{2},\phi_{D-1}) = -\mathbf{s}(\mathbf{V}_{N\times D};-\phi_{1:D-3},\frac{\pi}{2},\phi'_{D-1}), \forall\phi'_{D-1}\in\Phi$

(v) $\mathbf{s}(\mathbf{V}_{N\times D};\phi_{1:D-3},\pm\frac{\pi}{2},\phi_{D-1}) = \mathbf{s}(\mathbf{V}_{N\times D};-\phi_{1:D-3},\pm\frac{\pi}{2},\phi'_{D-1}), \forall\phi'_{D-1}\in\Phi$

for any $\phi_1,\phi_1,\ldots,\phi_{D-1}\in\Phi$, where $\mathbf{s}(\mathbf{V}_{N\times D};\phi_{1:D-1})$ is a binary vector mapped to the point $\phi_{1:D-1}$. If the intersection of $D-1$ hypersurfaces is uniquely defined then the vector $\boldsymbol{\phi}_{inters}$ that holds the spherical coordinates of that intersection leads a cell $C_{\phi_{inters}}$ (associated with a binary vector $\mathbf{s}_C$) that is above it, in the sense that $\boldsymbol{\phi}_{inters}$ is the single point of $C_{\phi_{inters}}$ for which ϕ_{D-1} is minimised. We collect all the above binary vectors in a set $\mathcal{J}$ whose size is $|\mathcal{J}| = \binom{N}{D-1}$. In other words, set $\mathcal{J}$ contains $|\mathcal{J}| = \binom{N}{D-1}$ binary vectors, each associated with a cell in Φ^{D-1} that minimises its ϕ_{D-1} component at the point given by the intersection of the corresponding $D-1$ hypersurfaces.

Using the equalities that hold true presented above, we can ignore the vectors that are associated with cells created for $\phi_{D-2}=-\frac{\pi}{2}$, set $\phi_{D-2}=\frac{\pi}{2}$, ignore ϕ_{D-1} and identify the cells that are determined by the reduced size matrix $\mathbf{V}_{N\times(D-2)}$ over the hypercube Φ^{D-3}. This means that the set $\mathcal{S}'$ of all possible vectors is:

$$\mathcal{S}'(\mathbf{V}_{N\times D}) = \mathcal{J}(\mathbf{V}_{N\times D})\cup\mathcal{J}(\mathbf{V}_{N\times(D-2)})\cup\cdots\cup\mathcal{J}(\mathbf{V}_{N\times(D-2\lfloor\frac{D-1}{2}\rfloor)}) \tag{3.19}$$

whose cardinality is shown to be $|\mathcal{S}'| = \sum_{d=0}^{D-1}\binom{N-1}{d}$. To summarise, $D-1$ auxiliary spherical coordinates were used to partition the hypercube Φ^{D-1} into $|\mathcal{S}'|$ regions that are associated with the same number of distinct binary vectors. Finding s_{opt} now costs $\mathcal{O}(N^{D-1})$ upon construction of $\mathcal{S}'$. An efficient algorithm for the construction of $\mathcal{S}'$ is presented next.

According to (3.19) the construction of $\mathcal{S}'(\mathbf{V}_{N\times D})$ is reduced to the parallel construction of all $\mathcal{J}$ sets. $\mathcal{J}(\mathbf{V}_{N\times 1})$, $\mathcal{J}(\mathbf{V}_{N\times 2})$ and $\mathcal{J}(\mathbf{V}_{N\times 3})$ can be obtained with complexity $\mathcal{O}(N)$, $\mathcal{O}(N\log N)$ and $\mathcal{O}(N^2\log N)$ [1], [41], [45]. So what remains, is to describe a way of constructing $\mathcal{J}(\mathbf{V}_{N\times d})$ for $d>3$. Also the construction of J can be fully parallelised,

since each candidate vector can be produced independently for each set of intersecting hypersurfaces. So it is only needed we present a way for the computation of each $\mathbf{x}_c$.

Consider a certain value of $d \in \{3, 4, \ldots, D\}$ and a certain set of indices $I_{d-1} \subset \{1, 2, \ldots, N\}$ such that $d-1$ hypersurfaces $S(\mathbf{V}_{i_1,1:d}), S(\mathbf{V}_{i_2,1:d}), \ldots, S(\mathbf{V}_{i_{d-1},1:d})$ intersect at ϕ_{inters}. The cell $C_{\phi_{inters}}$ that is "led" by ϕ_{inters} is associated with the binary vector $\mathbf{s}_{C_{\phi_{inters}}}$. For the N elements of this binary vector we observe:

1. For any $l \in \{1, 2, \ldots, N\} - I_{d-1}$ the corresponding element of $\mathbf{s}_{C_{\phi_{inters}}}$ maintains its value at ϕ_{inters} hence is determined by $s(\mathbf{V}_{n,1:d}; \phi_{inters})$.

2. For any $l \in I_{d-1}$ the corresponding element of $\mathbf{s}_{C_{\phi_{inters}}}$ cannot be determined at ϕ_{inters}, but maintains its value at the intersections of the remaining $d-2$ hypersurfaces, hence is determined by $s\ (\mathbf{V}_{n,1:d};\ \phi(\mathbf{V}_{N\times d-1};\ I_{d-1} - l))$.

The above observations suggest that if $l \in \{1, 2, \ldots, N\} - I_{d-1}$ the corresponding hypersurface $S(\mathbf{V}_{n,1:d})$ does not pass through ϕ_{inters} implying that the polarity of ϕ_{inters} with respect to $S(\mathbf{V}_{n,1:d})$ is the same as the polarity of any point in the cell $C_{\phi_{inters}}$ with respect to the same hypersurface. This means that the sign of the corresponding element of the binary vector $\mathbf{s}_{C_{\phi_{inters}}}$ is "well-determined" at the leading point. On the other hand, if $l \in I_{d-1}$ then an ambiguity arises concerning the corresponding element $s_{n;C_{\phi_{inters}}}$. This ambiguity is resolved if we exclude $S(\mathbf{V}_{n,1:d})$ and consider the intersection of the remaining $d-2$ hypersurfaces at $\phi_{d-1} = \frac{\pi}{2}$. The polarity of any point in cell $C_{\phi_{inters}}$ with respect to $S(\mathbf{V}_{n,1:d})$ is the same as that of $\phi(\mathbf{V}_{N\times d-1};\ I_{d-1} - l))$ with respect to the same hypersurface. Thus the sign of the corresponding element of the binary vector is well defined through point 2. above.

The last step that need describing is how the vector of the spherical coordinates of the intersections is computed efficiently. Recall that ϕ_{inters} represents the intersection of $S(\mathbf{V}_{i_1,1:d}), S(\mathbf{V}_{i_2,1:d}), \ldots, S(\mathbf{V}_{i_{d-1},1:d})$, i.e. the solution of

$$\mathbf{V}_{I_{d-1},1:d}\mathbf{c}(\boldsymbol{\phi}_{1:d-1}) = \mathbf{0}_{(d-1)\times 1}. \tag{3.20}$$

The above equation has a unique solution ϕ_{inters} which consists of the spherical coordinates of the zero right singular vector of $\mathbf{V}_{I_{d-1},1:d}$. This solution yields the coordinates of the intersections we are seeking. The complete MATLAB code for the construction of $\mathcal{S}'(\mathbf{V}_{N\times D})$ is provided below.

```
function X=compute_candidates(V)

% X=compute_candidates(V) returns a matrix whose columns are the
% corresponding binary vector candidates x for the maximisation of
% x'*V*V'*x

[N,D]=size(V);

if D>2
    combinations=nchoosek(1:N,D-1);
    X=zeros(N,size(combinations,1));
    for i=1:length(combinations)
        I=combinations(i,:);
        VI=V(I,:);
        c=find_intersection(VI);
        c=c*determine_sign_par(c);
        X(:,i)=sign(V*c);
        for d=1:D-1
            c=find_intersection(VI([1:d-1 d+1:D-1],1:D-1));
            c=c*determine_sign_par(c);
            X(I(d),i)=sign(VI(d,1:end-1)*c);
        end
    end
    X=[X compute_candidates(V(:,1:D-2))];
elseif D==1
    X=sign(V);
else
    phi_crosses=atan(-V(:,2)./V(:,1));
    [phi_sort,phi_ind]=sort(phi_crosses);
    X(phi_ind, 1:N+1)=(repmat(-sign(V(phi_ind,1)),[1 N+1]))...
     .*(2*tril(ones(N,N+1))-1);
end
end
```

```
-------------------------------------------------------------------------


function c=find_intersection(V)


% c=find_intersection(V) returns the zero right singular vector of V


[junk1, junk2, C]=svd(V);
c=C(:,end);


-------------------------------------------------------------------------


function sign_c=determine_sign_par(c)


D=length(c);
phi=zeros(D-1,1);
for phi_ind=1:D-1
    phi(phi_ind)=asin(c(phi_ind)/prod(cos(phi(1:phi_ind-1))));
end
if (phi(D-1)*c(D-1))==0
    sign_c=1;
else
sign_c=sign(tan(phi(D-1))*c(D-1)*c(D));
end
```

Complexity Analysis

Recall that the cardinality of $\mathcal{S}'$ is $\mathcal{O}(N^{D-1})$. This means that the same amount of intersections will be calculated and visited so as to compute the resulting candidate binary vector. The calculation of the zero right singular vector of $\mathbf{V}_{I_{d-1},1:d}$ costs $\mathcal{O}(d^2)$ while the conversion into spherical coordinates $\mathcal{O}(d)$. The operation $sign(\mathbf{V}_{n,1:d}\mathbf{u})$ costs $\mathcal{O}(d)$ for any $\mathbf{u} \in \mathbb{R}^d$ Since $\mathbf{u'} \in \mathbb{R}^{d-1}$ is computed for each $l \in I_{d-1}$, the cost of the procedure for each I_{d-1} is $\mathcal{O}(d^2) + (N-d+1)\mathcal{O}(d) + (d-1)(\mathcal{O}(d^2) + \mathcal{O}(d)) = O(d^3 + Nd) = O(N)$, since we have fixed $D \leqslant N-1$. Hence the overall complexity of the algorithm becomes $\mathcal{O}(N^{D-1})\mathcal{O}(N) = \mathcal{O}(N^D)$.

Chapter 4

Our approach

In this section we will discuss the algorithm we developed to solve (1.6). The approach we took is based on the work presented in [41]. We built on the ideas presented there and created a scalable algorithm which can be used to solve (1.6) independently of the rank of $\mathbf{A}$. Below we present and discuss the solutions for $\mathbf{A}$ matrices of rank 2, 3 and 4.

We start by presenting the solution for $D = 2$ for two reasons. Firstly it the most straightforward of the three cases to use as a base for explaining the conceptual building blocks of our algorithm. Secondly, the procedure used for the solution of the rank 2 case, will be implemented as a *base* when we describe the procedure for solving the $D = 3$ case. It follows that the rank-3 solution will be the base for the solution of the $D = 4$ case.

4.1 The rank-2 case

In this instance matrix $\mathbf{V}$ is a $N \times 2$ matrix while the ϕ_l in (3.18) define N points on the domain $\boldsymbol{\phi} \in [-\frac{\pi}{2}, \frac{\pi}{2})$ of the 1-dimensional space. ϕ_n divide $\boldsymbol{\phi}$ into $N + 1$ cells, each of which is defines a binary vector. The collection of these vectors is the set $\mathcal{S}'$ above is which we will search for s_{opt}. We continue this section by presenting a pseudo-code for the algorithm that solves this case and by conducting a step-by-step examination of its functions.

Pseudo-code 1.

1. Compute and sort into a set $\mathcal{F}$ in ascending order the values ϕ_n for all $n = 1, 2, \ldots, N$.
2. Set $\mathbf{s}_{opt} = \mathbf{s}_{\mathcal{F}(N)+1} = sign(\mathbf{V}_{1:N,1})$ and calculate its metric according to (1.6).

3. Set the above metric as the winning one and the $\mathcal{F}(N)+1$ binary vector as the optimal one.

4. For $i = \mathcal{F}(N-1) : -1 : \mathcal{F}(1)$ do:

   - Produce $\mathbf{s}_i$ by changing the sign on the $\mathbf{s}_{opt}(\mathcal{F}(i))$ entry.
   - Compare their respective $metric_i$ to the current $winner\_metric$.
     If $metric_i > winning\_metric$:
     $winning\_metric = metric_i$
     $winner = i$.

5. $i = \mathcal{F}(N-1) : -1 : winner$ do:

   - Produce $\mathbf{s}_i$ by changing the sign on the $\mathbf{s}_{opt}(\mathcal{F}(i))$ entry.

6. Return $\mathbf{s}_{opt}$.

Step 1 lies at the heart of this procedure. By calculating and sorting in ascending order the values ϕ_n, we effectively partition Φ^1 into $N+1$ segments (i.e. cells), each of which is described by a unique binary vector $\mathbf{s}_i$ that characterises the relationship of the cell with the points ϕ_n according to (3.17). In Step 2 we create the first binary vector we will utilise as the *base* for our iterative process to produce all $\mathbf{s}_i$. $\mathbf{s}_{\mathcal{F}(N)+1}$ describes the segment $[\phi_{\mathcal{F}(N)+1}, \frac{\pi}{2})$, where $\mathcal{F}(N)+1$ points to the ϕ_n with the largest value. Since this is the first binary vector we create, we also assign its value to s_{opt} and set a pointer to its position (i.e. $N+1$) to denote it as the optimal one.

The main concept of the algorithm is to break down the identification and the production of the optimal binary vector into two separate operations. The first time we visit each cell our purpose it twofold. First we produce the binary vector that characterises it and second we calculate the new metric in order to compare it with the one kept as the optimal (i.e. of largest value). This is done by adding twice the value $(-\mathbf{V}^T_{n,1:D}\mathbf{s}_{\mathcal{F}(N)+1}(\mathcal{F}(i)))$ to the previous metric. If the new metric is larger that the (temporary) optimal one, the new one is saved as optimal and the iteration in which it was found saved. This way, we are able to produce all $N+1$ metrics, identify the metric of the optimal candidate and save its position.

Producing the candidate vector for each new cell is a simple task once we take into account the knowledge from (3.18). Since the hypersurface (in this instance a point),

partitions the domain into two regions with opposite values $s_l(\phi) = \pm 1$, all that is needed is to "flip" the sign of the appropriate entry in the last produced candidate. The entry whose sign needs to be changed is denoted by the index i of the sorted vector that holds ϕ_n, where i is the value of the current iteration.

Now we have identified *where* the optimal candidate vector is, our last action is to produce and return it. During its second iteration over the cells, our algorithm begins with $\mathbf{s}_{\mathcal{F}(N)+1}$ and its sole purpose is to produce the next candidate vector up to the point where the first run identified the largest metric. Once the i-th candidate vector $\mathbf{s}_i$ has been produced, the execution ends and $\mathbf{s}_i$ is returned as the optimal solution.

Complexity Analysis

Step 1 is the most computationally demanding part of the algorithm as the ordering of N values takes time $\mathcal{O}(N\log N)$. Step 2 has both time and space complexities of $\mathcal{O}(N)$. The calculation of the metric for each new binary vector has fixed cost $\mathcal{O}(1)$, since the amount of computations required are proportional to D which in this case is equal to 2, resulting in $\mathcal{O}(D)$. Fixed are also the cost and memory requirements of Step 3, while Step 4 and 5 have time complexity $\mathcal{O}(N)$ since only one entry is altered during each iteration (during both steps) and the process of writing the winners is $\mathcal{O}(1)$. Note that step 5. adds the need for $\mathcal{O}(N)$ space to store an extra vector of size $N \times 1$, since we must have the original $\mathbf{s}_{\mathcal{F}(N)+1}$ to make the iterations a second time. Step 6. adds no extra space complexity since the iterative changes take place on a single vector. So, our algorithm returns the optimal candidate vector with time complexity $\mathcal{O}(N\log N)$ using $\mathcal{O}(N)$ space.

4.2 The rank-3 case

The solution for the $D = 3$ case follows the same basic principles as the one in the previous section, though some key differences do arise. Here we consider an input of a matrix $\mathbf{V}$ of size $N \times 3$. The hypersurfaces defined in (3.18) are *curves* on $\mathcal{D}_1 \times \mathcal{D}_2$, where $\mathcal{D}_1 \triangleq [-\frac{\pi}{2}, \frac{\pi}{2}]$ and $\mathcal{D}_2 \triangleq [-\frac{\pi}{2}, \frac{\pi}{2})$. Figure 3.1 that follows is an example that will help illustrate the above and serve as a reference in some aspects of the analysis that follows. Note that in this example we have matrix $\mathbf{V}$ of size 5×3.

The $N = 5$ curves partition $\mathcal{D}_1 \times \mathcal{D}_2$ into $\left(\frac{N(N+1)}{2} + 1\right) = 16$ cells that correspond to the same amount of candidate binary vectors. Each curve corresponds to a $\phi_n(\boldsymbol{\phi}_{2:D-1})$

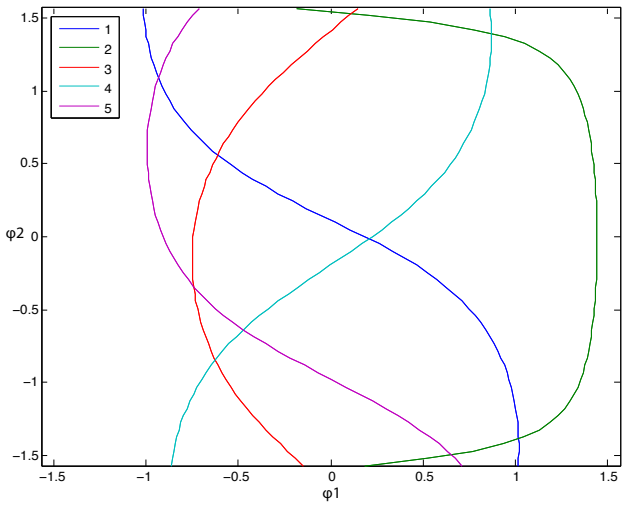


Figure 4.1: Partitioning of $\mathcal{D}_1 \times \mathcal{D}_2$ into regions

which for $D = 3$ is defined as

$$\phi_n(\boldsymbol{\phi}_2) \triangleq \tan^{-1}\left(- \frac{V_{n,2}\sin(\phi_2) + V_{n,3}\cos(\phi_2)}{V_{n,1}} \right) \in \left(-\frac{\pi}{2}, \frac{\pi}{2} \right) \tag{4.1}$$

and as with the rank-2 case, partitions $\mathcal{D}_1 \times \mathcal{D}_2$ into two regions, each of them associated with a unique $s_n(\phi_1, \phi_2) = \pm 1$. Searching among the $\left(\frac{N(N+1)}{2} + 1\right)$ binary vectors will yield the one that maximises the quadratic form.

The first and most important difference we perceive is that in the 2-dimensional space is that the hypersurfaces $\phi_n(\boldsymbol{\phi}_2)$ are now curves (instead of points) that intersect. [41] shows that each pair $\phi_l(\boldsymbol{\phi}_2), \phi_j(\boldsymbol{\phi}_2),\ l \neq j$ intersect only once and that the solution to $\phi_l(\boldsymbol{\phi}_2) = \phi_j(\boldsymbol{\phi}_2)$ is given by

$$\boldsymbol{\phi}_2(l, j) \triangleq \tan^{-1}\left(- \frac{V_{l,1}V_{j,3} - V_{j,1}V_{l,3}}{V_{l,1}V_{j,2} - V_{j,1}V_{l,2}} \right). \tag{4.2}$$

In addition one of the following statements is true.

1. If $\phi_l(\boldsymbol{\phi}_2) < \phi_j(\boldsymbol{\phi}_2)$ for $\boldsymbol{\phi}_2 \in [-\frac{\pi}{2}, \boldsymbol{\phi}_2(l, j))$ then $\phi_l(\boldsymbol{\phi}_2) > \phi_j(\boldsymbol{\phi}_2)$ for $\boldsymbol{\phi}_2 \in (\boldsymbol{\phi}_2(l, j), \frac{\pi}{2})$.

2. If $\phi_l(\boldsymbol{\phi}_2) > \phi_j(\boldsymbol{\phi}_2)$ for $\boldsymbol{\phi}_2 \in [-\frac{\pi}{2}, \boldsymbol{\phi}_2(l, j))$ then $\phi_l(\boldsymbol{\phi}_2) < \phi_j(\boldsymbol{\phi}_2)$ for $\boldsymbol{\phi}_2 \in (\boldsymbol{\phi}_2(l, j), \frac{\pi}{2})$.

We collect all coordinate θ of the intersections between lines into the set

$$\Theta \triangleq \{\theta \in \mathcal{D}_2 : \exists\ j = 1, 2, \ldots, N,\ k = 1, 2, \ldots, N,\ j \neq k, \qquad \text{such that } \theta = \boldsymbol{\phi}_2(l, j)\}$$

and note that $|\Theta| \leqslant \binom{N}{2} = \frac{N(N-1)}{2}$. In [2] the authors have shown that each intersection θ is the *leading* point for the cell that lies *above* it, as we move in $\mathcal{D}_2$ from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. Note

also that the binary vector of the cell it describes (i.e. the cell above) is the same as the one of the cell *below*, altering only the sign of the entries $s_l(\phi_1, \phi_2)$ and $s_j(\phi_1, \phi_2)$ [41].

It follows from the above that provided we have a base of $N + 1$ binary vectors that characterise the cells for $\theta \in [-\frac{\pi}{2}, \theta_1)$[1], if we sort all elements of Θ in ascending order and visit each intersection we will produce the candidate binary vectors that characterise all the cells in $\mathcal{D}_1 \times \mathcal{D}_2$. We continue by presenting the pseudo-code (and its analysis) for the algorithm that solves the $D = 3$ case.

Pseudo-code 2.

1. Create and sort by ascending order all elements θ in set Θ.
2. Execute the rank-2 case for $\phi_2 = -\frac{\pi}{2}$, creating the base of N+1 candidate vectors.
3. Produce the $N + 1$ metrics for the above candidates and set the one with the highest value as the *winner* (optimal).
4. For $i = \Theta(1) : \Theta(end)$ do:
   - Find the common cell between the intersecting curves.
   - Locate the vector in the base that is connected to that cell and create the new candidate vector s_i.
   - Calculate the $metric_i$ of the new vector and compare it with *winner*.
     If $metric_i > winning\_metric$:
     $winning\_metric = metric_i$
     $winner = i$.
   - Update the associations (see below).
5. For $i = \Theta(1) : \Theta(winner)$ do:
   - Find the common cell and create the new candidate vector s_i.
   - Update the associations.
6. Return $\mathbf{s}(winner)$ as the optimal candidate.

As can be seen, the concepts of the solution to the rank-2 case remain, but as we will elaborate below, a series of new challenges (and solutions) arise. The procedure behind

[1] θ_1 is the element of Θ with the smallest value, i.e. the first intersection

Step 1. is the same as in the $D = 2$ case though this time we need to sort the intersections developed by (4.2). After this is done we must construct our *base* of $N + 1$ binary vectors so as to have the necessary build blocks to implement the changes brought forth by the intersecting of the curves $\phi_n(\boldsymbol{\phi}_2)$'s. To achieve this in Step 2, we modify the algorithm used in the previous case ($D = 2$) and execute it on the 1-dimensional space that arises when we set $\phi_2 = -\frac{\pi}{2}$. This enables us to produce and store all the candidates that arise, along with their respective metrics. We also need to save the position of the winning metric.

Recall now that in the $D = 2$ case the changes produced by the points ϕ_n were serial, i.e. by moving along Φ^1 all points would be visited in a successive fashion by the iterative process. This meant that all changes would be produced on *one* binary vector(the latest one produced), thus only one needed to be saved. Unfortunately the introduction the second dimension $\mathcal{D}_2$ nullifies the certainty of serial changes. In layman's terms, we do not know which of the $N + 1$ candidates of our base will by affected by the changes produced at the θ intersection. This also renders the determining of the optimal vector problematic. The solution to this is two-fold and is a key process of our algorithm for $D > 2$.

In order to identify which cell, and in turn candidate vector, was to be affected we created an association of "adjacency" between two sets: the curves $\phi_n(\boldsymbol{\phi}_2)$ and the cells that appear in $\mathcal{D}_1 \times \mathcal{D}_2$. In the $D = 3$ case exactly two cells are adjacent to any curve $\phi_n(\boldsymbol{\phi}_2)$ for any $\boldsymbol{\phi}_2 \in \mathcal{D}_2$.

Visiting an intersection

Let us now return to Figure 4.1 and focus on a single intersection, in this scenario between curves c_2 and c_3. We arbitrary number the cells 4, 5, 6 and 5'. As we can see $\phi_2(\boldsymbol{\phi}_2)$ (curve c_2) is associated to cell 4 and cell 5 while $\phi_2(\boldsymbol{\phi}_2)$ (curve c_3) with cell 5 and cell 6, for any $\theta < \boldsymbol{\phi}_2(l, j)$ (i.e below the intersection). As we move along $\mathcal{D}_2$ from $-\frac{\pi}{2}$ towards $\frac{\pi}{2}$ and cross the intersection, the common cell 5 gives its place to 5' and the associations of lines and cells *flip*.

For any $\theta > \boldsymbol{\phi}_2(l, j)$, c_2 is now associated with cells 5' and 6, while c_3 with 4 and 5'. In other words the common cell (5) is the one that is replaced by the new one, while the other two cells exchange associations. Using this knowledge, we can successfully create the new candidate vector linked with the newly created area 5' and correctly depict the associations of curves and cells between any two intersections. One last step is now needed to identify the position of the candidate that needs to be altered in the base of $N + 1$ vectors in order

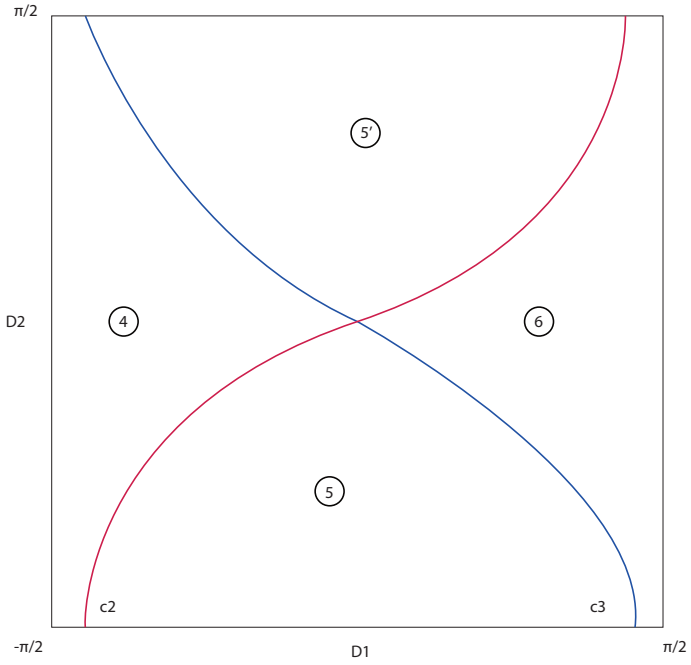


Figure 4.2: A closer look at an intersection

to produce the new one.

We create a vector of size $1 \times \left(\frac{N(N+1)}{2} + 1\right)$ and store pointers to the base cells in the first $N+1$ entries. In Step 4., for each iteration i over all the intersections θ we visit, we identify the common cell and place in the $(N+1)+i$-th entry the pointer to the position of the "original" one in the base. We then alter the binary vector placed there by changing the sign of the entries denoted by the two curves that intersect. In doing this, we have a structure that correctly describes the candidates after each intersection θ we visit. Finally we calculate (and compare to the winning one) the new metric based on the changes made to the previous one. Similarly to the rank-2 case the new metric is calculated by adding twice the values $\mathbf{V}_{l,1:D}^T s_l$ and $\mathbf{V}_{j,1:D}^T s_j$ to the previous one, where l, j the indexes of the curves c_l and c_j that intersect. If the new metric is larger, we save the iteration as the winner.

In Step 5. we follow the same iterative procedure, this time only producing the next binary vector up until the iteration we have saved as the winner in Step 4. Finally we return the last one produced as the optimal one.

Complexity Analysis

As with the previous case, Step 1. is the one that defines the overall complexity of the algorithm. Recall that $|\Theta| \leqslant \binom{N}{2} = \frac{N(N-1)}{2}$. This means that Step 1. has an overall complexity $\mathcal{O}(N^2 \log N)$ while the size of Θ gives us a space requirement of $\mathcal{O}(N^2)$. Step

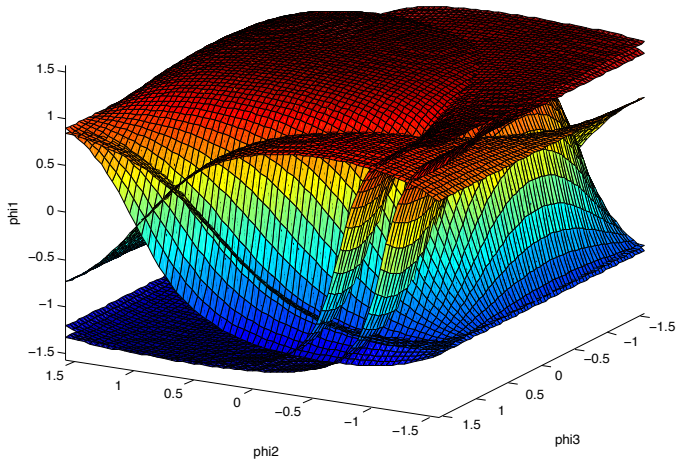


Figure 4.3: Partitioning of $\mathcal{D}_1 \times \mathcal{D}_2 \times \mathcal{D}_3$ into regions

2. introduces the structure with the largest memory requirement. The base we construct consists of $N+1$ binary vectors of size $N \times 1$ each, yielding an overall space requirement of $\mathcal{O}(N^2)$ while writing the entries can be completed in time of the same complexity. The vector that hold the metrics in Step 3. can be constructed in time $\mathcal{O}(N)$ and has a memory requirement of the same complexity. Steps 4. and 5. run for $\mathcal{O}(N^2)$ iterations with calculations of fixed complexity $\mathcal{O}(1)$ while the vector that holds the pointers to the base candidates has time and space requirements of $\mathcal{O}(N^2)$. Writing the changes of association is done in trivial time, while no additional memory is needed throughout these operations. The computation of the metrics is proportional to the rank $D = 3$ so are completed in fixed $\mathcal{O}(D)$ time.

This means that our approach operates within the theoretic bounds set in chapter 2 with time complexity $\mathcal{O}(N^{D-1}\log N)$ and space complexity of $\mathcal{O}(N^{D-1})$.

4.3 The rank-4 case

In this section we consider the case where rank($\mathbf{A}$) is $D = 4$. The idea of associating hypersurfaces and cells brought forth in the previous section are utilised once more, while the base of candidate vectors we need as the starting point of our algorithm is now produced by executing the procedure for the $D = 3$ case on the *bottom* of the 3-dimensional space, i.e. for $\phi_3 = -\frac{\pi}{2}$. The rank-3 algorithm executed there has also been modified with additions that will provide necessary data for the current procedure.

Figure 4.3 above demonstrates the partition of the 3-dimensional space described by $\mathcal{D}_1 \times \mathcal{D}_2 \times \mathcal{D}_3$ where $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3 \in \Phi^3$. A matrix $\mathbf{V}$ of size 5×4 was used to produce the

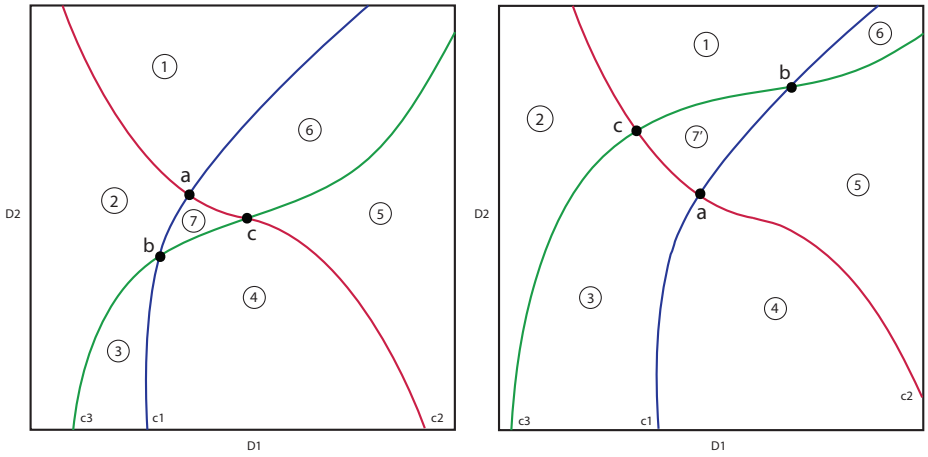


Figure 4.4: Before and after a triple intersection

figure.

Each $\phi_n(\boldsymbol{\phi}_2, \boldsymbol{\phi}_3)$ is given by

$$\phi_n(\boldsymbol{\phi}_2, \boldsymbol{\phi}_3) \triangleq \tan^{-1}\left(- \frac{V_{n,2}\sin(\boldsymbol{\phi}_2) + V_{n,3}\cos(\boldsymbol{\phi}_2)\sin(\boldsymbol{\phi}_3) + V_{n,4}\cos(\boldsymbol{\phi}_2)\cos(\boldsymbol{\phi}_3)}{V_{n,1}} \right) \quad (4.3)$$

and defines a surface (as seen in Figure 4.3) that partitions the 3-dimensional space into two regions with opposite $s_n(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \boldsymbol{\phi}_3) = \pm 1$. Following the reasoning used in the rank-3 case, we will move along $\boldsymbol{\phi}_3$ from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ searching for intersections of surfaces. In the $D = 4$ case though, only intersections of three surfaces at a time will produce a new cell and thus a new candidate binary vector. The authors in [2] show that these intersections are given by the spherical coordinates of the zero right singular vector of $\mathbf{V}_{I_{D-1}}$ where I_{D-1} denotes the subset of $D-1$ indices that correspond to the hypersurfaces that intersect. The set Θ that contains all coordinates θ of the triple intersection is of size $|\Theta| \leqslant \binom{N}{3} = \frac{N(N-1)(N-2)}{6}$. Therefore provided we construct a base including all the candidate vectors that exist for $\boldsymbol{\phi}_3 = -\frac{\pi}{2}$, then by visiting each triple intersection and computing the new candidate vector that occurs we only need to search among $\frac{N(N^2+5)}{6} + 1$ candidates to obtain the optimal one.

Visiting an intersection

In order to better explain the concepts of the process followed, we present figure 4.4 above that contains two plots. These are two *slices* of the 3-dimensional space and depict the space $\mathcal{D}_1 \times \mathcal{D}_2$ for $\boldsymbol{\phi}_3 = \phi_3(l, j, k) - e$ and $\boldsymbol{\phi}_3 = \phi_3(l, j, k) + e$, i.e. before and after a triple intersection.

As D has grown, it is no longer enough to use associations between curves and cells to locate the common one, in order to replace it with the new one that is produced. So we

introduce the concept of the **node** which denotes the intersections between two surfaces $\phi_n(\boldsymbol{\phi}_2, \boldsymbol{\phi}_3)$. Since the triple intersections concern three surfaces, it follows that the number of nodes produced by any single one is $\#nodes = \binom{3}{2} = 3$. Also from the definition of $\phi_n(\boldsymbol{\phi}_2, \boldsymbol{\phi}_3)$ we conclude that each node is associated with four cells.[2] This in turn leads us to the necessary additions we need to make to the algorithm use for the $D = 3$ case.

We introduce two new structures into the design of the rank-3 case. The first is a square matrix of size $N \times N$ and keeps the associations between any two curves and the node they produce, while the second is used to keep track of the associations of nodes and cells. We note at this point, that while the first structure once produced does not change, the associations between nodes and cells which are saved in the latter do, giving it a very dynamic nature. The process of initialising these structures is quite straightforward. During the execution of the rank-3 algorithm at each intersection we visit, we store in the appropriate entry of the former an index that corresponds to the intersection we are visiting, and at the corresponding index of the latter we store the cells that are associated with that intersection: the common one, the two "independent" ones and the new cell that is created. In doing so we form a link between hypersurfaces, nodes and cells.

Returning to figure 4.4 let us now present in pseudo-code form what procedure we follow once we visit a triple intersection to correctly compute the changes that occur.

Pseudo-code 3.

1. Retrieve the 3 nodes that are involved in the triple intersection.
2. For each node i=1:3 do:
    - In the association matrix do:
        - Replace common cell with the newly created one.
        - Exchange the rest associations (see below).

We identify the nodes we need to examine by knowing the three surfaces that intersect and looking up their relations to nodes in the first structure created above. Step 2. dictates we identify the common cell in the surface-to-cell associations. A visualisation of this can be seen in figure 4.4. The common cell (here number 7), lies "between" nodes a, b and c which means it is associated with all three nodes. The last step of the process tells us

[2] Each $\phi_n(\boldsymbol{\phi}_2, \boldsymbol{\phi}_3)$ divides the n-dimensional space into two distinct regions.

to update the associations so that when we reach the next triple intersection the correct ones will be stored.

Recall that in the $D = 3$ case this meant "exchanging" the associations of the areas that were only related to a single curve. In the current case we need a two step process in order to depict the correct associations after the intersection: remove the common cell and place the newly created one in its place and link up each node with all three cells it was *not* associated previously. This results in the following exchange for the example in figure 4.4:

Before	After
a → 1, 2, 6, 7.	**a** → 3, 4, 5, 7'.
b → 2, 3, 4, 7.	**b** → 1, 5, 6, 7'.
c → 4, 5, 6, 7.	**c** → 1, 2, 3, 7'.

Let us now return to the problem of finding the optimal candidate. After the exchange presented above is executed, the only thing that remains to be done is the correct identification and alteration of the candidate vector in our base, so that it expresses the new vector that is created. As with the respective step in the rank-3 case we create a vector of size $1 \times \frac{N(N^2+5)}{2} + 1$ and store pointers to the candidates in the base in the first $\frac{N(N+1)}{2} + 1$. Then by each time updating the correct entry with the pointer to the appropriate base candidate we have a structure that accurately represent the candidate vectors after each triple intersection. Calculating the metric of each new candidate is achieved the same way it was in the previous case. This time we need to add $2 * \mathbf{V}_{1:D}^T \mathbf{s}_{I_{D-1}}$ (three quantities), each one representing the change of the entry s_l given by the index of the surfaces that intersect.

We present in the form of pseudo-code the complete process for solving the rank-4 case.

Pseudo-code 4.

1. Create and sort by ascending order all triple intersections θ in set Θ.
2. Execute the rank-3 case for $\phi_3 = -\frac{\pi}{2}$, creating the candidate vector base.
3. Produce and store all metrics for the base and set the one with the highest value as the *winner* (optimal).
4. For $i = \Theta(1) : \Theta(end)$ do:

- Find the common cell between the intersecting surfaces.
- Locate the vector in the base that is connected to the common cell and create the new candidate vector s_i.
- Calculate its $metric_i$ and compare it with $winner$.
  If $metric_i > winning\_metric$:
  $winning\_metric = metric_i$
  $winner = i$.
- Run pseudo-code 3 in order to update associations.

5. For $i = \Theta(1) : \Theta(winner)$ do:
   - Find the common cell and create the new candidate vector s_i.
   - Run pseudo-code 3 in order to update associations.
6. Return $\mathbf{s}(winner)$ as the optimal candidate.

Complexity Analysis

The steps presented in pseudo-code 2 for the $D = 3$ case are the same we follow in our approach for the solution of the $D = 4$ one, so we use the same notation for our analysis here. Step 1., the computation and sorting of the triple intersections is accomplished in time $\mathcal{O}(N^3 \log N)$, since $|\Theta|$ is of size $\mathcal{O}(N^3)$. In Step 2. we execute the algorithm for the rank-3 case on the base ($\phi_3 = -\frac{\pi}{2}$) of our 3-dimensional space to obtain a base of candidate vectors. This structure consists of $\left(\frac{N(N+1)}{2} + 1\right)$ candidates of size $1 \times N$ and requires both time and space complexities of $\mathcal{O}(N^3)$ to fill.

Parallel to this we construct the vector saving the metrics of the base candidates. From the size of the set $|\Theta_{rank-3}|$ (recall this is $\mathcal{O}(N^2)$) and the fact that the computations required to determine a metric are of fixed $\mathcal{O}(D)$ complexity, this step is completed in $\mathcal{O}(N^2)$ while requiring space of the same amount. Again the amount of iterations in steps 4. and 5. is given by the size of Θ, which means that we complete $\mathcal{O}(N^3)$ iterations with a series of calculations (e.g. writing the new associations) with computational complexity of $\mathcal{O}(1)$. Finally we note again that the computation of the metric is fixed (proportional to $D = 4$) and thus costs $\mathcal{O}(D)$ for each iteration, not altering the overall cost of the procedure. The iterations in step 5. cost even less since only the new candidate is produced up until the one denoted as the winner.

The above shows that we have successfully created an algorithm that solves (1.6) for a rank $D = 4$ matrix in time $\mathcal{O}(N^3 \log N)$ using space of $\mathcal{O}(N^3)$. Aggregating all the results brought forth by our proposal of a serial algorithm utilising spherical coordinates, we have shown that our method is executed within the theoretic bounds of $\mathcal{O}(N^{D-1}\log N)$ and $\mathcal{O}(N^{D-1})$ for time and space complexity respectively.

4.4 Expanding for any rank-D

In this section we will set the foundations for a serial algorithm that solves the general case $D \geqslant 2$. The procedure of calculating the intersections and sorting them in ascending order will, of course, remain the same. This is also the most costly step of the algorithm since the set Θ that holds all the intersections of will always be of size $\binom{N}{D-1}$, i.e. $\mathcal{O}(N^{D-1})$, thus their sorting operation yields a complexity of $\mathcal{O}(N^{D-1}\log N)$. Also, the execution of the rank $D-1$ case at the base of the respective problem does not effect the overall complexity, as it always has complexity of $\mathcal{O}(N^{D-2}\log N)$.

Moving on, the the double iteration over all intersections also remains unaltered. One new candidate binary vector will occur for each new cell that is created by an intersection and it will be calculated by changing the sign of the entries that correspond to the $D-1$ hypersurfaces that intersect. These changes are also responsible for the altering of the previous metric in order to gain the new one. Following the reasoning in the previous chapters we need to add the $D-1$ values $2 * s_n \mathbf{V}_{n,1:D}$ of the respective intersecting hypersurfaces.

All that remains, is to describe a generalised way of creating the necessary framework to save and correctly update the associations between the *types* describing the intersections (e.g. points, curves, nodes etc.) and the cells that surround them. By visiting the cases 2, 3 and 4 that we previously analysed we see a pattern forming. In the rank-2 scenario, the hypersurfaces are points on a 1-dimensional plane while for the rank-3 case they form curves. The intersection is between $D-1=2$ curves each associated with two cells, one of which they have in common. Increasing the rank to the $D=4$ case, our analysis showed that we needed to define $D-1=3$ nodes which take part in the intersection. These nodes fully describe a 2-dimensional region which is the new cell that is created. To sum up a new candidate vector is created as follows:

- Rank $D=2$ a single point divides the space into two discrete regions.
- Rank $D=3$ two curves create a single point, the intersection which produces a new

vector.

- Rank $D = 4$ three nodes (i.e. intersections above) create an area which is responsible for the new vector.

If we continue expanding, it follows that for $D = 5$ four areas, i.e. the ones defined in the $D = 4$ case, will surround a 3-dimensional region (the new cell) directly associated with the new candidate. By observing the above, we see that for any rank D, we have $D-1$ intersecting hypersurfaces that create $D-2$ structures (or types) that are immediately correlated with the resulting intersection. Also note that each of these structures describes one *whole* intersection of the $D-1$ case. By finding the common area and following the procedures in the $D = 3$ and $D = 4$ cases for "exchanging" associations between the nodes, areas, 3-dimensional regions etc. one has all the tools one needs for the creation of the algorithm.

Below we present a pseudo-code which could act as a basis for the construction of the code that solved the general rank-D case.

Pseudo-code 5.

1. Create and sort by ascending order all triple intersections θ in set Θ.
2. Execute the rank-$(D-1)$ case for $\phi_{D-1} = -\frac{\pi}{2}$, creating the candidate vector base.
3. Produce and store all metrics associated with the base vectors,compare to find and set the one with the highest value as the $winner$.
4. For $i = \Theta(1) : \Theta(end)$ do:
   - Find the common cell between the $D-1$ intersecting hypersurfaces.
   - Visit the vector in the base that is connected to the common cell and create the new candidate vector $\mathbf{s}_i$ by *flipping* the $D-1$ signs that correspond to the intersecting hypersurfaces.
   - Calculate its $metric_i$ and compare it with $winner$.
     If $metric_i > winning\_metric$:
     $winning\_metric = metric_i$
     $winner = i$.
   - For all $D-2$ structures/types do:

- Replace the common area with the newly created one.
- "Exchange" the rest associations.

5. For $i = \Theta(1) : \Theta(winner)$ do:
   - Find the common cell and create the new candidate vector s_i.
   - For all $D - 2$ structures/types do:
     - Replace the common area with the newly created one.
     - "Exchange" the rest associations.
6. Return $\mathbf{s}(winner)$ as the optimal candidate.

If all the above observations are taken into account, then future studies could focus on the creation of a unified serial procedure for the solution of any $rank(A) = D \geqslant 2$ case. Another focus would be the optimisation of certain aspects of the algorithm, in order to eliminate any unnecessary space or time redundancies in the procedures involving secondary variables.

Chapter 5

A recursive approach

In this chapter we take a closer look at the algorithm proposed in [44]. The authors propose a solution, taking a recursive approach to solving (1.6). In the following we investigate this solution by analysing the proposed algorithm and study the complexities of each step in order to confirm the short analysis presented in chapter 2.

5.1 Theoretic Developments

Using the matrix $\mathbf{V}$ defined in chapter 1 we can define N hyperplanes in $\mathbb{R}^D : H_j = \{\mathbf{a} \in \mathbb{R}^D | V_j \cdot a = \sum_{i=1}^{D} a_i v_{ij} = 0\},\ j \in \{1, 2, \ldots, N\}$. Notice now that there exists a one-to-one correspondence between the vectors in $\{-1, 1\}^N$ for which there exists a vector $a \in \mathbb{R}^D$ such that $s_j = sign(\sum_{i=1}^{D} a_i v_{ij})$, with $\sum_{i=1}^{D} a_i v_{ij} \neq 0\ \forall\ j \in \{1, 2, \ldots, N\}$ and the cells in (i.e. full dimensional regions) in $\mathbb{R}^D$ of the hyperplane arrangement $\mathcal{A}(H)$ defined by the hyperplanes $(H_j)_{j=1}^{N}$. This is easier to grasp if we interpret the sign vector $\mathbf{s}$ as the position vector of its corresponding cell with respect to an orientation of the space by the vector $\mathbf{V}_j$. The cell is *above* the hyperplane H_j iff $s_j > 0$ and *under* otherwise. For a general arrangement in $\mathbb{R}^D$ that is defined by L hyperplanes, the number of cells is upper-bound by $\sum_{i=0}^{D} \binom{N}{i}$, which is $\mathcal{O}(N^D)$. Since in our case, all hyperplanes contain the origin (i.e. the arrangement is central), this is reduced to $\mathcal{O}(N^{D-1})$ [39].

Lets start by defining $I^+ = \{i \in \{1, \ldots, N\} | s_i = 1\}$ and $I^- = \{1, \ldots, N\} \backslash I^+$. A vector $\mathbf{s} \in \{-1, 1\}^N$ will correspond to a polyhedron that is defined by $\{a \in \mathbb{R}^D | V_i \cdot a \leqslant 0,\ \forall\ i \in I^-$ and $V_i \cdot a \geqslant 0\ \forall\ i \in I^+$. Recall that the cells in $\mathcal{A}(H)$ correspond exactly to the full dimensional polyhedra of the latter type. The vectors $\mathbf{s}_j$ that correspond to the cells are defined in the following manner: starting with an initial vector $\mathbf{s}^0 \in \{-1, 1\}^N$ and $\mathcal{L}$

a chained list of indexes $Ind \in \{1, \ldots, N\}$ that correspond to the variables s_i that are multiplied by -1, we use the first subset of indices Ind in $\mathcal{L}$ and create $\bar{\mathbf{s}} \in \{-1, 1\}^N$ such that $Ind = \{i \in \{1, 2, \ldots, N\} |\ \mathbf{s}_i \neq \bar{\mathbf{s}}_i\}$. We then replace $\mathbf{s}^0$ with $\bar{\mathbf{s}}$ and iterating over all the elements in $\mathcal{L}$, we create a set of vectors in $\{-1, 1\}^N$ that fully describe the polyhedra in $\mathcal{A}(H)$ mentioned above.

5.2 Algorithmic Developments

Two procedures were developed in [44]. The first deals with the case $D = 2$ while the second with the general case for $D \geqslant 2$ utilising the first as a base.

The case $D = 2$

We start by observing that since the arrangement is central, if a vector $\mathbf{s} \in \{-1, 1\}^N$ represents a cell, so does the vector $-\mathbf{s}$. Thus, in order to determine all vectors associated with cells it suffices to know only the ones that are on one side of the hyperplane, since the rest can be produced by the component-wise opposite vectors.

Lets now assume that we have an $N \times 2$ matrix $\mathbf{V}$ and we want to determine the full dimensional polyhedra in the arrangement $\mathcal{A}(H)$ they define. We start by finding those that are *above* the hyperplane H_1, by considering the arrangement that is defined by $H_2, \ldots H_N$ in the affine subspace $H_1' = \{a \in \mathbb{R}^2 | V_1 \cdot a = 1\}$, which is of dimension 1. The cells of this arrangement have extremities that are either unbounded or correspond to an intersection $H_1' \cap H_j$ for some $j \in \{2, \ldots, N\}$.

These intersection are now determined as follows: by using the equation that defines H_1' (and assume w.l.o.g. that $v_{2,1} \neq 0$) we eliminate the variable a_2 from the system of equations $V_j \cdot a = 0$, thus obtaining a new system $\overline{V}_j \cdot a_1 = b_j$ where $\overline{V}_j, b_j \in \mathbb{R}^2$ and $j \in \{2, \ldots, N\}$. We then create a new set $F = \left\{ (\frac{b_j}{\overline{V}_j}, j) \mid j \in \{2, \ldots, N\} \right\}$. This is properly defines since the assumption of non-collinearity of rows implies that $\overline{V}_j \neq 0$.

Now we order the values in F and we note that these define N intervals in $\mathbb{R}$. Note also that any cell in $\mathcal{A}(H)$ that is above H_1 has a point a, with a_1 belonging to any such interval. Assume that Int is an interval. Then for any $a_1 \in\ Int$ we either have $\overline{V}_j \cdot a < 0$ or $\overline{V}_j \cdot a > 0$ for any $j \in \{2, \ldots, N\}$ and this relation remains unchanging in Int. It follows that Int corresponds to a cell in H_1' and the converse also holds.

Algorithmically speaking, after ordering the elements of F by increasing value we proceed with the following. First we create the initial vector s^0 that corresponds to the interval $\left(-\inf, \min \frac{b_j}{\overline{V}_j} \right)$ by setting:

- $\mathbf{s}_1^0 = 1$
- $\mathbf{s}_j^0 = -sign(\overline{V}_j)\ \forall\ j \in \{2, \ldots, N\}$.

The final step is the creation of the chained list $\mathcal{L}$ so that all cells will be represented. The first elements of the lists are the $N-1$ indexes j of the ordered set F. We add the element $\{1, \ldots, N\}$ to the end of the list $\mathcal{L}$. This will create a binary vector that represents the first cell on the opposite side of H_1'. Finally we add to the list, all the elements of $\mathcal{L}$ except the last one (i.e. $\{1, \ldots, N\}$) but in reverse order. This will ensure that we will crate the binary vectors necessary to represent all cells in the arrangement $\mathcal{A}(H)$.

The case $D \geqslant 2$

The basic principle behind the procedure for $D \geqslant 2$ is the following: for an integer $q \in \{1, \ldots, N\}$, $\mathcal{A}^q(H)$ is the arrangement in the subspace $\{a \in \mathbb{R}^D \mid V_q \cdot a = 0\}$ that is defined by the hyperplanes (in $\mathbb{R}^{D-1}$) $H_j \cap H_q \mid j \in \{1, \ldots, N\},\ j \neq q\}$. Any cell (a region of dimension D-1) in $\mathcal{A}^q(H)$ is a facet of two cells c_1 and c_2 of $\mathcal{A}(H)$, created in the $D = 2$ case. We conclude that since each cell of $\mathcal{A}(H)$ intersects at least one hyperplane H_q, by varying the value of q we will produce all cells in the arrangement at least once.

The proposed algorithm recursively calls itself while reducing the size (and thus the rank D) of the input matrix $\mathbf{V}$ until it reaches the case where $D = 2$. Then it calls the procedure generated above and builds upon the the results returned. A more in-depth look at the algorithm reveals the following.

If the rank of the input matrix $\mathbf{V}$ is 2, then apply the procedure presented above. Else for each $i \in \{1, \ldots, N\}$ we express the system of equations $V_j \cdot a = 0,\ j \in \{1, \ldots, N\},\ j \neq i, a \in \mathbb{R}^D\}$ in the subspace $H_i = \{a \in \mathbb{R}^D |\ V_i \cdot a = 0\}$. By doing this, removing row i and some column, we result in a new system of equations $\overline{V} \cdot \overline{a} = 0$ where $\overline{V} \in \mathbb{R}^{(N-1)\times(D-1)}$ and $\overline{a} \in \mathbb{R}^{D-1}$. We then make a recursive call to the procedure with the matrix $\overline{V}$ as the input. Each of the equations of the new system $\overline{V} \cdot \overline{a} = 0$ defines a hyperplane of the subspace H_i. The recursive calls result in a description $(\bar{\mathbf{s}}_i^0, \overline{\mathcal{L}_i})$ of the cells in H_i that are induced by the hyperplanes $H_i \cap H_j,\ i \neq j$.

After the description $(\bar{\mathbf{s}}_i^0, \overline{\mathcal{L}_i})$ has been produced, we must express $\bar{\mathbf{s}}_i^0$ in $\mathbb{R}^N$ by adding one entry set to 1. This is placed according to the index i (removed from V when producing $\overline{V}$ using the equation defining H_i). The next step is to edit the linked list $\overline{\mathcal{L}_i}$ so as to represent the changes. For each subset of indexes Ind that is represented in $\overline{\mathcal{L}_i}$ we do the following:

- Increment by one all indices in Ind that are greater than or equal to i.
- Insert a set of indexes containing only $\{i\}$ before Ind.

The addition of a singleton $\{i\}$ after each element that is already in the list representing a cell c in H_i, is used to create two cells: one above and another below H_i in the original space, each of them having c as a facet. Moreover, because the description of cells in H_i corresponds to a set of vectors in $\mathbb{R}^{D-1}$ we are required to adjust them for correct representation in $\mathbb{R}^D$ by adding one additional entry corresponding to the i-th component of the description vectors in the original space. Last but not least we must adjust the existing indices as follows: if $j < i$ then the j-th row of the system $\overline{V} \cdot \overline{a} = 0$ corresponds to the j-th hyperplane of the original system $V \cdot a = 0$. But if $j > i$ then the j-th row of $\overline{V} \cdot \overline{a} = 0$ corresponds to the $j + 1$-th row of the original system. Thus, indices greater of equal to i in the list $\overline{\mathcal{L}_i}$ must be incremented by one.

The next steps involve the unification of all $(\bar{\mathbf{s}}_i^0, \overline{\mathcal{L}_i})$ into a single output. Firstly we need to add to the end of each list $\overline{\mathcal{L}_i}$ a set Ind that contains the positions at which the last position vector of $(\bar{\mathbf{s}}_i^0, \overline{\mathcal{L}_i})$ differs from the vector $\bar{\mathbf{s}}_{i+1}^0$. To achieve this we must iterate over all elements of all sets Ind in $\overline{\mathcal{L}_i}$ in order to achieve the production of the last position vectors.

After this modification has been done for all $i \in \{1, \ldots, N-1\}$ we can set $\mathbf{s}^0 = \mathbf{s}_1^0$ and concatenate $\overline{\mathcal{L}_1}, \ldots, \overline{\mathcal{L}_N}$ (in this order) into the single list $\overline{\mathcal{L}_D}$ so that all position vectors that were in separate lists, are now in a single output. The procedure then returns $(\bar{\mathbf{s}}^0, \overline{\mathcal{L}_D})$ as a complete representation of the set that contains the solution to the problem.

After obtaining $(\bar{\mathbf{s}}^0, \overline{\mathcal{L}_D})$ procuring $\mathbf{s}_{opt}$ is a matter of iterating over all elements in $\overline{\mathcal{L}_D}$ and calculating the metric produced by the newly created position vectors. The method followed is the same presented in our approach in chapter 3.

Complexity Analysis

In order to verify the time and space complexity costs, as well as the size of the set $|\mathcal{S}|$ presented in chapter 2, an analysis of the algorithm is presented below. For $D = 2$ the total number of indices that are in the output list of any call to the procedure is $\mathcal{O}(N)$. Also the total space required for the creation of the sets F and $\mathcal{L}$ is $\mathcal{O}(N)$ since their size is proportional to N.

Moving on to the procedure for $D \geqslant 2$ we can point out that the number of indices contained in each list $\overline{\mathcal{L}_i}$, increases by $|\overline{\mathcal{L}_i}| + 1$, with $|\overline{\mathcal{L}_i}|$ is the total number of *sets* of indices contained in $\overline{\mathcal{L}_i}$. This is $\mathcal{O}(N^{D-2})$ and by concatenating the lists in order to

produce $\overline{\mathcal{L}_D}$ we result in having a list containing $\mathcal{O}(N^{D-1})$ indices. Another point we must pay attention to in the search for the entries that differ between the last position vector of $(\bar{\mathbf{s}}_i^0, \overline{\mathcal{L}_i})$ and $\bar{\mathbf{s}}_{i+1}^0$. By utilising the space needed for a single position vector and iterating over the indexes in $\overline{\mathcal{L}_i}$ while applying the necessary changes on $\bar{\mathbf{s}}_i^0$, we manage to keep the space complexity within the set boundaries.

As for the time complexity of the procedure we deduce the following: in the procedure for the case $D = 2$ expressing each equation $V_j \cdot a = 0$ in the affine subspace $H_1' = \{a \in \mathbb{R}^2 | V_1 \cdot a = 1\}$ and creating the initial vector $\mathbf{s}^0$ needs $\mathcal{O}(N)$ time. The addition of the sets of indexes to $\mathcal{L}$ at the end of the procedure also needs $\mathcal{O}(N)$ time, which can be deduced from the space complexity analysis above. The most costly part of the procedure is the creation and ordering of the set F. For each $j \in \{2, \ldots, N\}$ we compute the values $\frac{b_j}{V_j}$ and order them. The latter takes times $\mathcal{O}(N\log N)$.

In the case where $D \geqslant 2$ each recursive call take $\mathcal{O}(N^{D-2}\log N)$ time while from the space complexity analysis above we recall that the total number of indices in any list $\overline{\mathcal{L}_i}$ is $\mathcal{O}(N^{D-2})$. By utilising the procedure for all $i = \{1, \ldots, N\}$ we conclude that the total time needed for the creation of the set $(s^0, \mathcal{L}_D)$ and the returning of s_{opt} is $\mathcal{O}(N^{D-1}\log N)$.

Chapter 6

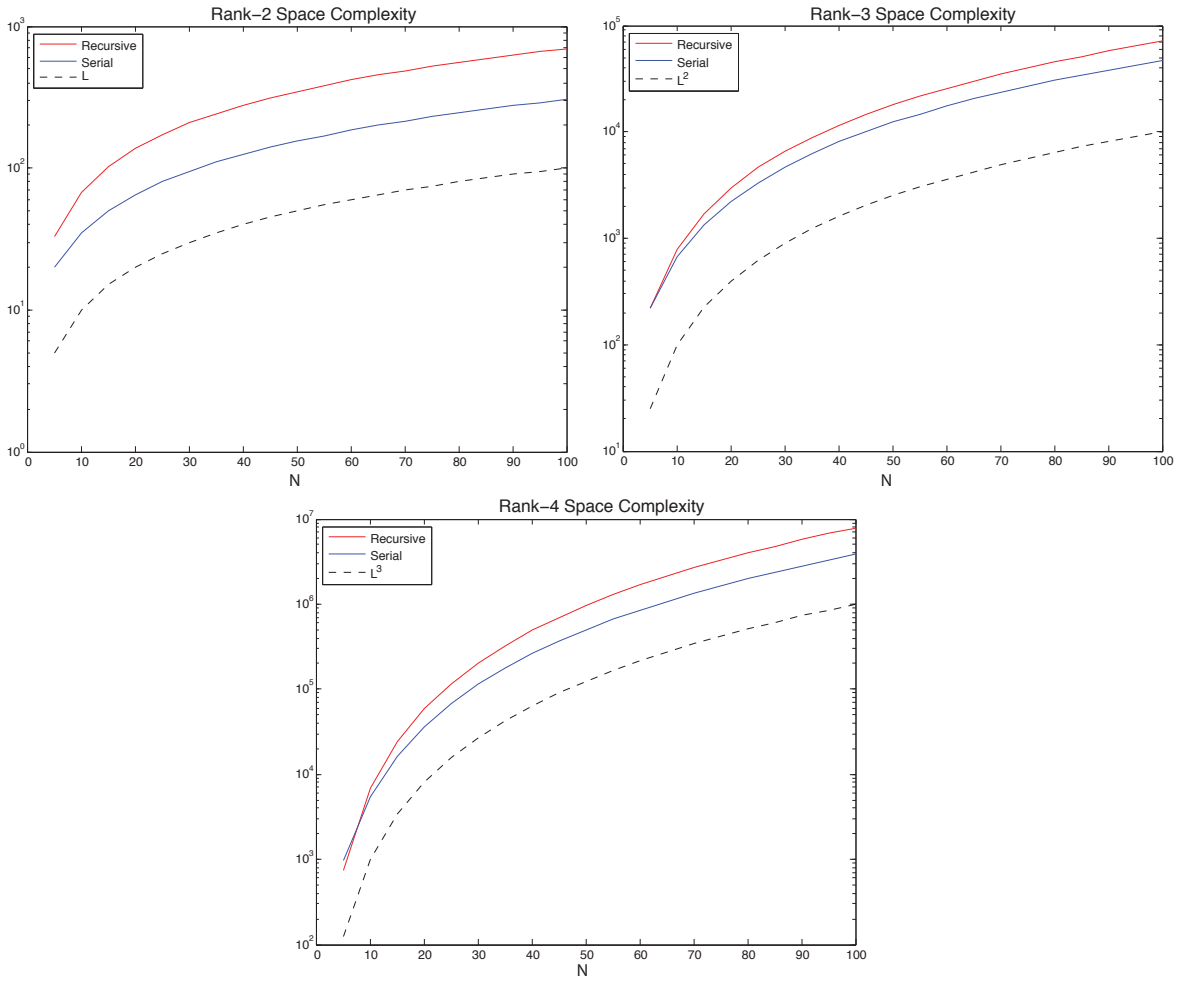
Comparisons

The focus of this chapter is the comparison of the actual time and space complexities of the serial algorithm that utilises auxiliary spherical coordinates we developed and the one proposed in [44] and described in the previous chapter. Recall in chapter 2 the table that presents the summary of approaches for the solution of (1.6). By looking at the table we see that the two approaches have the same theoretic time and space complexities, $\mathcal{O}(N^{D-1}\log N)$ and $\mathcal{O}(N^{D-1})$ respectively. Furthermore, both approached produce a set of candidate binary vectors $|\mathcal{S}|$ of size $\mathcal{O}(N^{D-1})$.

Despite the same theoretic complexities, differences do arise when the procedures are simulated. The algorithms were coded in MATLAB and simulated in the same environment, while the computations were performed using double precision. The simulations were run on a 2.3 GHz Intel Core i5 with 8GB of RAM. In the diagrams that follow we illustrate the *actual* time and space complexities of the two procedures for ranks 2, 3 and 4 as N grows.

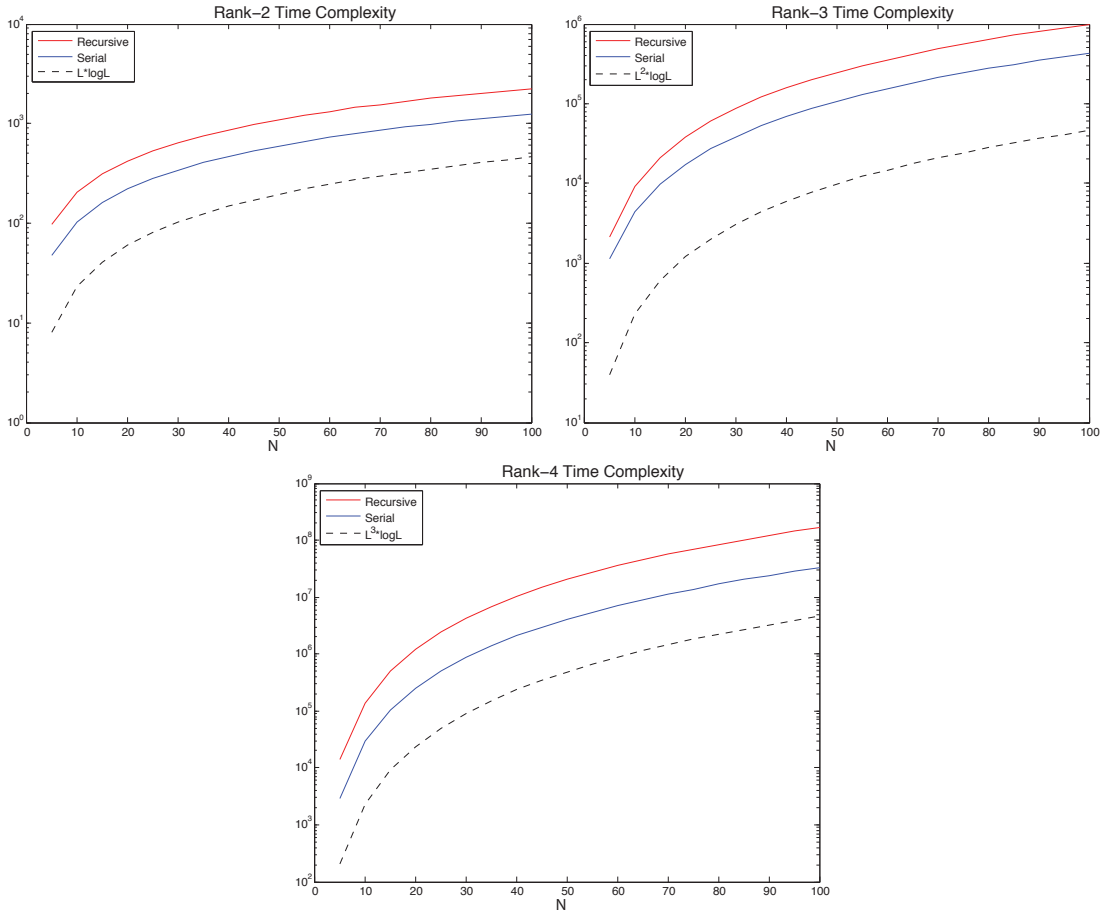
The figures below are plotted using a logarithmic axis y (vertical axis). In all figures in addition to plotting the comparison data, we also include as a dotted line the respective theoretic bound, so as to confirm that our algorithmic data follows the same slope as N grows.

In the first set of figures we present the space complexities of the the two procedures and are able to draw some interesting conclusions. The first thing we must note is that both approaches do follow the slope of the theoretic bound $\mathcal{O}(N^{D-1})$ thus also proving their practical adherence to it. Secondly we clearly see that the serial algorithm using auxiliary coordinates we developed, has a smaller memory requirement that the one proposed in [44]. Focusing on this and by taking a closer look at the results for small cases of N, in the rank 3 and especially in the rank 4 results, we can view another interesting finding.



Due to a larger amount of secondary variables needed in our approach, we see that our proposal starts with a slightly larger memory requirement (i.e. $N \cong 5$ for the rank 3 and $N \cong 7$ for the rank 4 case). This is quickly overshadowed though by the overall space requirements of the procedures as can be seen as N increases.

The next set of figures presents the actual time complexities of the two procedures. As with the case of the space complexity plots above, we also plot the theoretic bound $\mathcal{O}(N^{D-1}\log N)$ (as a dotted line) to study their adherence to it.



The situation here is more clear-cut. Both approaches follow the curve produced by the theoretic bound but our algorithm is consistently below the recursive approach one. This means that less overall calculations are used in our serial approach utilising the auxiliary spherical coordinates.

An explanation for the appearance of the figures above can be sought in the size $|\mathcal{S}|$ of the set of binary vectors each algorithm produces. While [44] even for the rank-2 case produces a set whose size is substantially larger than N (i.e. $2N$) the size of the set $\mathcal{S}$, only gets increasingly larger (bound by $\mathcal{O}(N^{D-1})$) as D increases. Recall that for every call of the recursive function the size of each $\overline{\mathcal{L}_i}$ is increased by $|\overline{\mathcal{L}_i}|+1$. On the other hand our implementation produces $N+1$ binary vector candidates for the $D=2$ case while for $D=3$ and $D=4$ we have a total of $\left(\frac{N(N+1)}{2}+1\right)$ and $\left(\frac{N(N^2+5)}{6}+1\right)$ respectively. We note that the multiplier of the N^{D-1}-th element always has a divisor, so it follows that the size $|\mathcal{S}|$ of our candidates set is always a fraction of N^{D-1} and thus our set smaller. This, in turn, leads to a smaller amount of computations being needed, as can be seen in the figures above. □

Bibliography

[1] G. N. Karystinos and D. A. Pados, "Rank-2-optimal adaptive design of binary spreading codes," *IEEE Trans. on Inf. Theory, vol. 53, pp. 3075-3080*, Sept. 2007.

[2] G. N. Karystinos and A. P. Liavas, "Efficient computation of the binary vector that maximizes a rank-deficient quadratic form," *IEEE Trans. Inf. Theory, vol. 56, pp. 3581-3593*, July 2010.

[3] Barahona, F., Groetsel, M., Junger, M., Reinelt, G.: "An application of combinatorial optimization to statistical physics and circuit layout design", *Oper. Res. 36, 493-513*, 1988

[4] Groetsel, M., Junger, M., Reinelt, G.: "Via minimization with pin preassignmets and layer preference", *J. Appl. Math. Mech./ Zeitschrift fur Angewandte Mathematik und Mechanik 69, 393-399*, 1989.

[5] Pinter, R. Y., "Optimal layer assignment for interconnect." *Adv. VLSI Comput. Syst. 1, 123-137*, 1984.

[6] Heath, R.W. Jr, Paulraj, A.: "A simple scheme for transmit diversity using partial channel feedback" in *Proceedings of IEEE Asilomar Conference on Signals, Systems and Computers vol. 2, pp. 1073-1078*, Pacific Grove, November 1998.

[7] Kim Y.G., Beaulieu, N.C.: "On MIMO beamforming systems using quantizer feedback" *IEEE Trans. Commun. 58, 820-827*, 2010.

[8] Leung, K.-K., Sung, C.W, Khabbazian, M., Safari, M.A.: "Optimal phase control for equal-gain transmission in MIMO systems with scalar quantization: complexity and algorithms" *IEEE Trans. Inf. Theory 56, 3343-3355*, 2010.

[9] Love D.J., Heath, R.W. Jr, Strohmer, T.: "Grassmanian beamforming for multiple-input multiple-output wireless systems", *IEEE Trans. Inf. Theory 49, 2735-2747*, 2003.

[10] Ryan D.J., Clarkson I.V.L., Collings, I.B., Guo, D., Honig, M.L.: "QAM and PSK codeblocks for limited feedback MIMO beamforming", *IEEE Trans. Commun. 57, 1184-1196*, 2009.

[11] Santipach, W., Mamat, K.: "Tree-structured random vector quantization for limited-feedback MIMO beamforming" *IEEE Trans. Commun. 57, 1184-1196*, 2009.

[12] Xu, M., Guo, D., Honig, M.L., "MIMO precoding with limited rate feedback: simple quantizers work well" in *Proceedings of IEEE GLOBECOM 2009*, Honolulu, December 2009.

[13] Zheng, X., Xie, Y., Li, J., Stoica, P.: "MIMO transmit beamforming under uniform elemental power constraint", *IEEE Trans. Signal Process. 55, 5395-5406*, 2007.

[14] B. Hassibi, H. Vikalo, "On the sphere-decoding algorithm I. Expected complexity", *IEEE Trans. Signal Proc. 53, 2806-2818*, Aug. 2005.

[15] H. Vikalo, B. Hassibi, "On the sphere-decoding algorithm II. Generalizations, second-order statistics, and applications to communications", *IEEE Trans. Signal Proc. 53, 2819-2834*, Aug. 2005.

[16] S. Verdu, "Multiuser detection", *New York, NY: Cambridge University Press*, 1998.

[17] G. Manglani and A. K. Chatuverdi, "Application of computational geometry to multiuser detection in CDMA", *IEEE Trans. Commun. 54, 204-207*, Feb. 2006.

[18] L.Brunel, "Multiuser detection techniques using maximum likelihood sphere decoding in multicarrier CDMA systems", *IEEE Trans. Wireless Commun. 3, 949-957*, May 2004.

[19] E. Viterbo, E. Beglieri, "A universal decoding algorithm for lattice codes", *Proc. Quatorzieme Colloque GRETSI, 611-614*, 1993.

[20] A. Banihashemi, A. Khandani, "On the complexity of decoding lattices using the Korkin-Zolotarev reduced basis", *IEEE Trans. Inf. Theory, 44, 162-171,* Feb. 1998.

[21] E. Agrell, T. Eriksson, A. Vardy, K.Zeger, "Closest point seach in lattices", *IEEE Trans. Inf. Theory 48, 2201-2214*, Aug. 2002.

[22] P.D. Alexander, A.J. Grant, M.C. Reed, "Iterative detection in code-division multiple-access with error control coding", *Eur. Trans. Tel. 9, 419-425*, 1998.

[23] A. Stefanov, T.M. Duman, "Turbo-coded modulation for systems with transmit and receive antenna diversity over block fading channels: System Model, decoding approaches and practical considerations", *IEEE J. Select. Areas Commun. 19, 958-968*, May 2001.

[24] B. Hcowald, S. ten Brink, "Achieving near-capacity on a multiple antenna channel", *IEEE Trans. Commun. 51, 389-399*, Mar. 2003.

[25] B. Hassibi, B.M. Hochwall, "High-rate codes that are linear in space and time", *IEEE Trans. Inf. Theory 48, 1804-1824*, June 2002.

[26] M.O. Damen, A. Chkeif, J.-C. Belfiore, "Lattice code decoder for space-time codes", *IEEE Commun. Lett. 4, 161-163*, May 2000.

[27] A. Y. -C. Peng, L.-M. Kim. S. Yousefi, "Low complexity sphere decoding algorithm for quasi-orthogonal spacetime block codes", *IEEE Trans. Commun. 54, 377-382*, Mar. 2006.

[28] W.H. Mow, "Maximum likelihood sequence estimation from the lattice viewpoint", *IEEE Trans. Inf. Theory 40, 1591-1600*, Sept. 1994.

[29] A. Alavi, C. Tellambura, I. Fair, "PARP reduction of OFDM signals using partial transmit sequence: An optimal approach using sphere decoding", *IEEE Commun. Lett. 9, 982-984*, Nov. 2005.

[30] T. Chui, C. Tellambura, "Joint data detection and channel estimation for OFDM systems", *IEEE Trans. Commun. 54, 670-679*, April 2006.

[31] W. Zhao, G. B. Giannakis, "Sphere decoding algorithms with improved radius search", *IEEE Trans. Commun. 53, 1104-1109*, July 2005.

[32] H. Artes, D. Seethaler, F. Hlawatsch, "Efficient detection algorithms for MIMO channels: A geometrical approach to approximate ML detection", *IEEE Trans. Sign. Proc. 51, 2808-2820*, Nov. 2003.

[33] K. Allemand, K. Fukuda, T. M. Liebling and E. Steiner, "A polynomial case of unconstrained zero-one quadratic optimisation," *Mathematical Programming, vol. A-91, pp.49-52*, Oct. 2001.

[34] J.-A. Ferrez, K. Fukuda and T.M. Liebling, "Solving the fixed rank convex quadratic maximisation in binary variables by a parallel zonotope construction algorithm," *European Journal of Operational Research, vol. 166, pp. 35-50*, 2005.

[35] D. Avis and K. Fukuda, "Reverse search for enumeration," *Discrete Applied Mathematics, vol. 65, pp. 21-46*, Mar. 1996.

[36] N. Meggido, "Linear programming in linear time when the dimension is fixed," *J. ACM., vol. 31, pp. 114-127*, Jan. 1984.

[37] I. Motedayen, A. Krishnamoorthy and A. Anastasopoulos, "Optimal joint detection/estimation in fading channels with polynomial complexity," *IEEE Trans. Inf. Theory, vol.53, pp. 209-223*, Jan. 2007.

[38] H. Edelsbrunner, J. O'Rourke and R. Seidel, "Constructing arrangements of lines and hyperplanes with applications," *SIAM J. Comput., vol. 15, pp. 341-363*, May 1986.

[39] H. Edelsbrunner, *Algorithms in combinatorial geometry*, New York: Springer-Verlag, 1987.

[40] K. M. Mackenthun, Jr., "A fast algorithm for multiple-symbol differantial detection of MPSK," *IEEE Trans. Commun., vol. 42, pp. 1471-1474*, Feb./Mar./Apr. 1994.

[41] G. N. Karystinos and A. P. Liavas, "Efficient computation of the binary vector that maximizes a rank-3 quadratic form," in *Proc. Allerton Conf. Commun., Control, and Computing, Allerton House, Monticello, IL, Sep. 2006*, pp. 1286–1291.

[42] A. Kyrillidis and G. N. Karystinos, "Fixed-rank Rayleigh quotient maximisation by an MPSK sequence," *IEEE Trans. Commun.*, to appear.

[43] K.-K. Leung, C. W. Sung, M. Khabbazian and M. A. Safari, "Optimal phase control for equal-gain transmission in MIMO systems with scalar quantisation: Complexity and Algorithms", *IEEE Trans. Inf. Theory, vol. 56, pp. 3343-3355*, July 2010.

[44] W. Ben Ameur and J. Neto, "A polynomial-time recursive algorithm for some unconstrained quadratic optimisation problems," *Discrete Applied Mathematics, vol. 159, pp.1689-1698*, Sept. 2011.

[45] G. N. Karystinos and D. A. Pados, "Rank-2 optimal binary spreading codes", in *Proc. 2006 Conf. on Inform. Sc. and Syst. (CISS 2006), Princeton Univercity, Princeton, NJ, pp. 1534-1539*,Mar. 2006.